Master Thesis

Department of Informatics

Faculty of Mathematics and Natural Sciences

University of Oslo

Model-based Secure Software Engineering using UMLsec applied to Assisted Living and Home Care

by

Kudakwashe Kidwell Chambwe
kudakwac@student.matnat.uio.no

Under the supervision of

Christian Johansen
&
Olaf Owe

2018

Abstract

With the emergence of Internet of Things (IoT), there is growing number of interconnected devices being developed with concerning security vulnerabilities. Consequently we are experiencing attacks and breaches that are capable of doing significant damage. And because security can often be difficult to properly enforce, consume time and result in higher development costs, we shall propose and examine the UMLsec approach. The approach is aimed at improving the development of secure systems. Meanwhile, new technologies and methods and approaches for developing systems like the ThingML are surfacing, that are aimed at being cost-effective, less time-consuming and increase the productivity. Therefore during the course of this thesis we shall attempt to bridge these two approaches and address the security challenges we face today.

We introduce the concept of Model-Driven Secure Software Engineering (MDSSE) for specifying and enforcing security requirements at UML design in order to enforce established rules of prudent security engineering early in the software development process. We define a process of defining and constructing a UML profile, using the provided extension mechanisms to extend the UML metalanguage with security concepts and well-formedness rules by [Jürjens, 2005] and UMLsec. To demonstrate and validate the approach, we define and propose the ThingMLsec profile which extends the security concepts and threat model of UMLsec for the domain of IoT systems. This approach and demonstration is supported by the Eclipse-based UML2 modeling tool Papyrus and related extensions.

In order to demonstrate and validate our ThingMLsec profile, we use the use case and scenario of Assisted Living and Community Care provided by the project Secure COnnected Trustable Things (SCOTT). We show that this highly expressive, effective and applicable approach, combined with a wide variety of proficient tools can help system engineers, developers and designers to specify and automatically verify security requirements in developing for the Internet of Things or any other domain.

Until lambs become lions...

Acknowledgments

I would like to take this opportunity to express my eternal gratitude to everyone who have supported me. Not only throughout the course of this thesis, but during the course of my Masters degree.

I am deeply grateful to my main supervisor Christian Johansen. This thesis would not be possible without your guidance and direction. I would also like to acknowledge my second supervisor Olaf Owe, for your valuable and insightful comments on this thesis.

Finally, I would like to thank my family and friends. Specifically my loving mother and stepfather, for encouraging, supporting and believing in me. Thank you so much.

With warm regards,

Kudakwashe K. Chambwe

Contents

List of Figures

List of Tables

List of Acronyms

ALCCS Assisted Living and Community Care System. 3, 5, 8–10, 18, 62, 66, 68–72, 79

API Application Programming Interface. 23

BLE Bluetooth Low Energy. 8, 9, 31, 32

BTL Binary Temporal Logic. 84

D2D Device-to-Device. 6–8, 31

DAC Discretionary Access Control. 15

DMSL Domain Specific Modeling Language. 4, 12, 35, 55, 59, 86, 90, 93

DoS Denial of Service. 6, 32

ECA Event-Condition-Action. 23

ECDH Elliptic Curve Diffie Hellman. 32

FIFO first in, first out. 23

GDPR General Data Protection Regulation. 92

IoT Internet of Things. i, 2, 4–6, 10, 79, 93

LAN Local Area Network. 26, 28, 30, 31, 33

LTL Linear Temporal Logic. 84

MAC Mandatory Access Control. 15

MBSE Model-Based Security Engineering. 4, 13, 18, 25, 79

MDSE Model-Driven Software Engineering. 4, 10, 12, 13, 18, 21, 58, 90

MDSSE Model-Driven Secure Software Engineering. i, 5

MDT Model Development Tool. 55

MITM Man-in-the-Middle. 6, 31, 32

OCL Object Constraint Language. 15–17, 55, 58

OMG Object Management Group. 58

OO Object-Oriented. 25

PET Privacy Enhancing Technologies. 91, 92

POS Point-of-Sale. 30

RBAC Role-Based Access Control. 15, 72

SCOTT Secure COnnected Trustable Things. i, 3, 8, 9, 62, 74, 79

TLS Transport Layer Security. 91

UML Unified Modeling Language. 13

VPN Virtual Private Network. 30, 91, 92

WDA Wireless Data Aggregator. 66–69

WPAN Wireless Personal Area Network. 8

Part I

Introduction

CHAPTER 1

Introduction

1.1 Motivation

The underlying foundation for the idea we shall explore is interconnectivity; the idea that everything, from our watch to the light bulb inside our living rooms, can somehow be remotely controlled from an application, device or gadget of some-sort. This is a growing technological trend today in and has introduced what is called the "Internet of Things (IoT)". This trend, which is essentially an increase in connected devices, is not something new, but because of the growth of mobile devices, this has acquired so much interest amongst people lately. Certain technological trends like "Big Data", "Blockchain" etc. are now the driving forces for what seems to be the "Next Big Thing". But what exactly makes this so revolutionary? One cannot argue that the IoT is currently reshaping the world. Everything from elevators to airplanes. Essentially changing the way we live, work and play – everything from our personal health to the functioning of entire cities.

Major technological trends like "Big Data" are currently the driving forces behind the emergence of IoT. However the true catalyst of IoT is the growing inexpensiveness of devices and their building blocks. It is therefore important to realize that this is – in many ways – a double-edged sword. Because while devices and their building blocks are cheap and quickly assembled, they are unfortunately highly insecure. Hence this is therefore an important issue to be addressed. Consequently the outcome at this stage is that an increasing number of companies desperately seek leverage from IoT as they hope to win big in this so-called "gold rush", that is IoT. For unlike some of largest companies like Apple, Amazon, Samsung etc. are constantly raising the bar in terms of their well established ecosystems of interconnected devices, more companies are desperately trying to keep up, in hopes to defy the odds and "win big" in the gold rush. However this is one of the main challenges that are leading to IoT devices with huge security flaws.

It goes without saying that the sheer scale and impact of IoT security has has on todays society, makes this issue paramount. As much as 30 billion devices are expected to be connected within the year 2020 [Tiwary et al., 2018]. Furthermore, these interconnected devices are everywhere from planes, elevators, power suppliers, health-critical devices (e.g. pacemakers), homes (e.g. baby monitors), cars, bank services etc. A study conducted by [Ponemon Institute LLC, 2017] showed that 80% of organizations do not routinely test their IoT applications for security vulnerabilities. Needless to say that this is relatively alarming as it enables malicious actors to spy, steal, disrupt, and ultimately endanger peoples lives with these applications. The correlation between the high number of devices currently in use, the indifference and relaxed approach that companies have in developing these devices, together with the ease of access malicious actors have on these devices. Another worrying aspect of this is that the devices are developed with an identical structure. This enables attackers to easily attack a high number of devices and further increasing the risk of IoT applications.

We currently see IoT appearing in many areas and many forms; whether it be in our homes (i.e. "Smart Home") or our public spaces (i.e. "Smart City"). Regardless of this, one of the most promising areas in terms of the appliance of IoT is within health services. Use of medical devices like sensors and wearables are helping saving lives, by simply being able to monitor and control drug doses, monitor patients, assisting medical personal and even monitoring vital signs (i.e. heart rate, breathing, skin temperature etc.). This is also the purpose of the Assisted Living and Community Care System (ALCCS) initiated by Secure COnnected Trustable Things (SCOTT), which has the intention of enabling elderly people to remain living in the comfort of their homes. This shall be the context for which we shall examine the applicability of a specific security-enforcing UML extension and approach called UMLsec.

Having established that the IoT can be hacked, manipulated and misused by malicious actors, there are various solutions and approaches for dealing with security for software development. The general consensus amongst many regarding this is that the earlier the treatment, the better. The approach of UMLsec proposes extending the general purpose language of UML to handle security at the specification and design level of the software development process. The idea is that for some organizations, security is sometimes not a top priority in IoT development and for other companies, that IoT applications are hard to properly secure. So our solution is propose using an approach that focuses on handling security early in the process, with the purpose to make software development easier and less costly for all parties involved. There is no doubt that this is not enough for a comprehensive handling of the security issues we face today, and more importantly in terms of the Internet of Things, but we do believe that this will ease the process of changing the perception of security sometimes being a burden or that it is hard to implement during development.

With increasingly new appliances of technologies today, we are seeing new methods and approaches to development attempting to cope with this evolution. These new methods are increasingly trying to reduce the costs, increase the efficiency and productivity in developing new systems. One specific approach that aims to achieve this is ThingML. ThingML (as in "things"/devices for IoT)is a Domain Specific Modeling Language (DMSL) and code generation framework aimed at generating code from models [Harrand et al., 2016]. The unique features of this framework is that it is specific to the domain of IoT, aims to be more customizable than other similar approaches and to provide the necessary abstraction that developers need to improve productivity. Since the language uses a combination of architecture models, state machines and an imperative action language, we hope to improve and validate this approach by integrating it with UMLsec. This attempt is to further improve the usability and applicability of both approaches within the domain of Internet of Things (IoT). The work in this paper is concerned with exploring how we can adopt and extend the UML extension and approach of UMLsec, to add and improve security for the code generation framework of ThingML, or eventually any Model-Driven Software Engineering (MDSE) approach or framework.

1.2 Problem Formulation

Our objective with this thesis is to specify and enforce security at a specification and design level, tailored explicitly for Model-Driven Software Engineering (MDSE). Using the UMLsec approach for Model-Based Security Engineering (MBSE), the idea is that we can apply security concepts early in the software development process and verify the models against such concepts to further improve the security in software development. This thesis will explore mainly three approaches:

- *Model-Driven Software Engineering*: concerned with development of systems based from an initial set of requirements and the model from it, to ultimately implement a system from that model.
- *Model-Based Security Engineering*: concerned with handling security at the specification and design level.
- *Using the UML extension mechanisms for extending the UML metalanguage*: concerned with using the UML extension mechanisms of *stereotype*, *tags* and *constraints* to extend UML concepts for domain-specific use.

By adopting these approaches, we will try to address the problem of security vulnerabilities in modern day software development (i.e. specific to the domain of IoT) which is also the motivation for this thesis. Therefore with regards to this matter, we will attempt to answer the following questions:

(i.) In terms of *usability*, how is the UMLsec approach?

(ii.) Is the UMLsec approach *applicable* to a real-world use case (i.e. SCOTT's Use Case of Assisted Living Community Care System)?

(iii.) Can the UMLsec approach be *combined* with the code generation framework of ThingML to facilitate Model-Driven Secure Software Engineering?

1.3 Contributions

During our work on this thesis, we have made various contributions and these are described more detailed below:

1. *A ThingMLsec profile for domain-specific security specification and design of Internet of Things (IoT) systems.*
   During Chapter 4, we introduce the ThingMLsec profile, that extends the UMLsec approach by [Jürjens, 2005] by adding new domain-specific security concepts for Model-Driven Secure Software Engineering (MDSSE) and security specification at UML design.

2. *A domain-specific threat scenario for modeling and analyzing possible adversary behavior within Internet of Things (IoT).*
   The basic threats from the *default* attacker specified in UMLsec, does not address the communication links threats for IoT technologies. During Chapter 3 we introduce a modified and more specific threat scenario with added domain-specific concepts for the *defaultIoT* attacker.

3. *A detailed process for any system designer or system engineer to define a profile, apply security requirements from that profile and verify the models against the constraints using the Papyrus modeling tool.*
   During the Chapter 4 we present the UML model elements and metaclasses corresponding each stereotype and associated tags for defining a profile, and finalize this by summarizing the UML2 diagrams corresponding to the applicable stereotypes. We proceed by demonstrating the process of applying the stereotypes, tags and constraints to various static and behavioral diagrams to eventually verify the models against the specified constraints.

4. *Application of the ThingMLsec profile and threat scenario of "defaultIoT" on the use case and scenario of Assisted Living and Community Care System (ALCCS).*
   During Chapter 6, we specifically apply our domain-specific ThingMLsec profile and *defaultIoT* threat scenario to a real-world use case of Assisted Living and Community Care. Consequently, this demonstrates that UMLsec and our ThingMLsec is applicable to any domain and use case, expressive enough to enforce most classes of constraints

on various levels, and lastly since most of artifacts and model elements are provided by UML, we can can use a wide variety of tools to replicate and improve upon this work.

1.4 The Internet of Things

Increasing intercommunication has paved the way for the emergence of Internet of Things (IoT) which is constantly changing the way we interact with the digital world in many ways. And with new "things" or intercommunicating objects, comes new opportunities and possibilities..

[Prehofer and Chiarabini, 2015] defines the idea of Internet of Things (IoT) as "[. . .] a variety of things or objects – such as RFID (Radio-Frequency IDentification), tags, sensors, actuators, mobile phones etc. – which are able to interact with each other and cooperate with their neighbors to reach a common goal. Our increasingly connected world is the result of these networks of "things" communicating and interacting with each other. Another notable development in this current evolution of IoT, is that while the number of things are constantly increasing, new businesses, services and other areas for this type of technology are growing silmutainiously. This is paving way for new and better ways of developing applications for IoT. The approach mostly discussed in this paper will be the model-based approach using ThingML[1].

1.4.1 Device-to-Device Communication for Internet of Things

Devices are an integral part of the IoT ecosystem. For it is the devices or smart objects/"things", that are are the actual users of IoT, and therefore the essential requirement for IoT is to provide connectivity between devices. The new Device-to-Device (D2D) Communication is therefore a vital part of IoT. D2D Communication can essentially be divided into cellular license spectrum like cellular communication (i.e. inband communication), and unlicensed bands like Wifi-Direct, Bluetooth, Zigbee and NFC (i.e. outband communication) [Militano et al., 2015]. An important issue to address in terms of D2D communcation is security and privacy, which is also one of the main topics for this thesis.

The reason for addressing D2D communication is that the direct wireless connection of D2D makes the connections highly vulnerable to security threats. Secure wireless connection can be achieved by satisfying the security requirements of authenticity, privacy, confidentiality, integrity and availability. This is required to successfully ensure and provide protection against common attacks Denial of Service (DoS), masquerading, eavesdropping, Man-in-the-Middle (MITM) attacks. Therefore central requirement for D2D communication is protection through some form of encryption. In the subsequent subchapters we will explain two essential D2D

[1] `http://thingml.org/pmwiki.php`

communication technologies for the IoT – 5G and Bluetooth. These technologies will be discussed further in the thesis in terms of the security vulnerabilities and possible solutions.

5G

The fifth and next generation (5G) of cellular systems is the response to the increasing demand for higher data rates and capacity and represents a new form of cooperative communications [Tehrani et al., 2014]. The conventional cellular systems, devices (i.e. portable devices with wireless connectivity) are not allowed to communicate directly with each other in the licensed cellular bandwidth and the communications take place through the base stations (BS). The next generation of cellular systems or 5G will allow for *device relaying* – enabling devices in a network to function as transmissions relays for each other and realize a large-scale ad-hoc mesh network. 5G will also, with the functionality of Device-to-Device (D2D) communication make it possible for two nearby devices to communicate with each other in a licensed cellular bandwidth with either limited to none BS involvement.

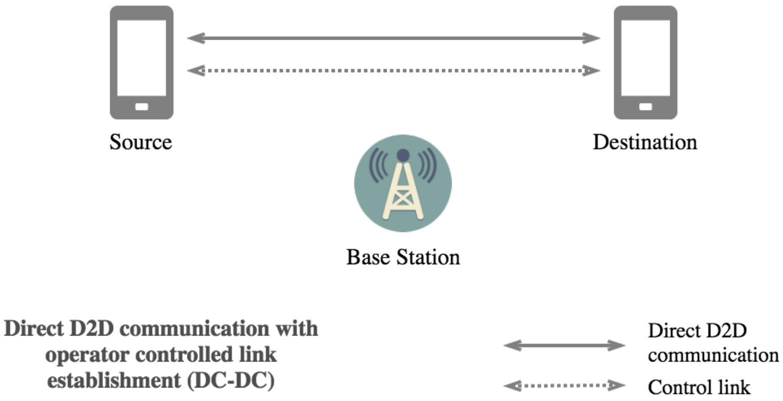

Source
Destination
Base Station
Direct D2D communication with operator controlled link establishment (DC-DC)
Direct D2D communication
Control link

Figure 1.1: D2D Communication with Device Controlled Link Establishment

5G cellular networks distinguishes between two types of two-tier cellular networks. (i.) Microcell tier (BS-to-device communication) and (ii.) device tier (D2D communication) [Tehrani et al., 2014]. For this thesis however, we will be focusing on the latter – two tier cellular network involving D2D communications. Nonetheless there exist 4 types of device tier communications, and from the use case description of SCOTT (i.e. physical view of ALCCS), we assume the communication labeled with 5G, is within Direct D2D communication, and specifically the Direct D2D communication with device controlled link establishment(DC-DC). The other alternative however is the Direct D2D communication with operator controlled link establishment (DC-OC). The reason behind this assumption is that the BS which would represent the operator, is not present in the physical view described in the scenario, making it natural to assume that the operator is non-existent. With this form of communication, the user's data is being sent form one device to the other, and therefore we will address the security and privacy issues in further detail in Chapter 3.

BLE

Bluetooth is an open standard for short-range radio frequency (RF) communication. It is a wireless technology used primarily to establish Wireless Personal Area Network (WPAN), and allows for a network between a wide range of devices to either transfer data or voice [Padgette et al., 2017]. Bluetooth Low Energy (BLE) represents a recent realization to meet the constraint for IoT devices and for the new form of application scenarios like remote monitoring involving BLE-enabled wearable sensors for the use of smartphone connectivity. This is central to the scenario that will be introduced in Chapter 6. Bluetooth like many D2D connection technologies are vulnerable to several wireless networking threats and vulnerabilities. Therefore to improve the security of both Bluetooth 5G we shall look at the preexisting and other ways of ensuring security in Chapter 3.

1.4.2 Use Case Application of Internet of Things

The application of UMLsec during this thesis will be for the Assisted Living and Community Care System (ALCCS) proposed by the project Secure COnnected Trustable Things (SCOTT) [Brandsma, 2017]. This use case enables frail elderly people (e.g. whom are prone to falling) to remain living in the comfort of their own home longer. This attempt is motivated by the incentive to lower the ever soaring costs of health care and aging population. The solution is a personal response system using multi-modal sensoric information to deduce the context of the elderly resident, as well as that of potential caregivers, to select the most appropriate caregiver to help. The Caregiver in this case, may be informal, which enables less costly and more trustful exploitation of the personal emergency.

A typical scenario for the use case of ALCCS is e.g. the "Call For Help" scenario which is one of the scenarios demonstrating the trust delegation within the context of emergency handling. The scenario specifically demonstrate how the resident call for help by activating or pressing the panic button and how the ALCCS handles the entire flow related to that specific situation. The "Call For Help" will be the use case mostly in focus in terms of demonstrating the modeling of behavior diagrams in relation to UMLsec.

1.4.3 Why the Use Case of Assisted Living?

There are many reasons why the Assisted Living and Community Care System (ALCCS) is perfect for the demonstration of UMLsec. In short the reasons are connected to the objectives specified by [Brandsma, 2017]. In this subsection we will go into the most relevant objectives to further clarify the motivation for choosing this use case.

i. Focus on Wireless Systems

The ALCCS uses in-home wireless connectivity like Wi-Fi, Bluetooth Low Energy (BLE), proprietary ones to communicate, localize and securely enter the given premises via proximity. The incentive will explore cutting-edge technologies to achieve low power, low bandwidth, low cost, high coverage cellular technologies to allow for more affordable, reliable, anytime, anywhere device connectivity to the device worn by the Resident. The chosen technologies will serve as the introduction of a wider palette of cellular technologies for IoT during the introduction of 5G (fifth generation) cellular networks.

The relevance of this objective is that the one of the frequently exploited areas in terms of security is the communication links. This also applies to IoT systems, and there the utilization of cutting-edge technologies within connectivity is highly relevant for the application of such a domain-specific approach like UMLsec. It is therefore interesting to explore the expressiveness and adaptability of UMLsec in terms of new technologies and communication links that did not exist at the time of its inception.

ii. Focus on Smart Sensor & Actuators

Another focus area of the ALCCS is using smart sensors and actuators to make the effort of caring and monitoring the elderly location and vital-signs etc. The SCOTT system specification for Assisted Living comprises specifically a Smart Lock as an example of a smart actuator and a variety of other state-of-the-art sensors. Furthermore the tasks of some of the smart sensors will be to able to locate the resident (i.e. elderly) in their own home. The proceeding iterations will consider various wearable vital-sign sensors, as well as ubiquitous localization.

Therefore since the main objective for the thesis is to adopt and demonstrate an approach to specify and enforce security requirements (i.e. UMLsec) at the design level of systems development (i.e. IoT domain), the requirement for the use case is fitting since it utilities sensors and actuators thus representing a suitable IoT ecosystem.

iii. Focus on Security, Safety, Privacy & Trustability

The edge-based (i.e. distributed) solution for the ALCCS is centered on trust delegation enabling an elderly person to automatically solicit help from Caregivers (e.g. family members) in a privacy-aware manner. The system takes care of all context derivation and trust-based delegation. Therefore any information about the "accidents" and people involved does not have to leave the premises unless the situation requires it so. Some security and privacy measures are incorporated into the infrastructure, like the choice of a focusing on a distributed computing mode as opposed to centralized, with also an option for mixed computing available as an option.

The strong and clear focus on security and privacy is one of the best reasons why the application of UMLsec to the ALCCS is so suitable. While privacy is not primarily the focus for this thesis, it is considered in the construction and application of the profile extension of UMLsec. With UMLsec we will try to improve exactly these focus areas, thus making the use case particularly suitable.

1.4.4 The Goal

The "Use Case Specification and System Architecture for Assisted Living" by [Brandsma, 2017] provides a multi-view architecture of the ALCCS. With this in mind, the objective for this thesis is therefore to enforce security requirements to the various views and diagrams provided by the use case. Since one of the questions that this thesis will try to answer is the applicability of UMLsec in a real-world use case, the objective with this effort is to compare the aforementioned diagrams and views, with the security-enforced UMLsec diagrams, in order to finally attempt to discuss the usability of the whole process. From this process, we will eventually discuss the possibility of adopting UMLsec to a specific approach of Model-Driven Software Engineering (MDSE) that is ThingML.

1.4.5 Developing for Internet of Things

To provide a more complete overview of the development of Internet of Things (IoT), we need to address the alternative methods and approaches currently in use. The heterogeneous nature of Internet of Things (IoT) is the main reason why we need different methods, tools and approaches to develop IoT devices and software. [Prehofer and Chiarabini, 2015] mentions the two most common approaches and tools for developing applications for IoT today. These are IoT Mashup tools and Model-based approaches (e.g. ThingML).

Mashup Tools

Mashup tools are visual, interactive modeling tools for the message flow between devices. The main idea behind mashup tools is to provide a simple way to develop IoT applications, and this is done by mashing up existing services in the Web. Well known mashup tools are *WoTKit*, *Clickscript* and *Paraimpu* [Prehofer and Chiarabini, 2015].

Model-Based Approaches

The other approach to be addressed and which is relevant for this thesis is the model-based approach using *ThingML*. This specific model-based framework will be discussed in further details in section 2, but the main concept behind model-based development with ThingML is to model and generate code using the DSL (Domain Specific Language) and framework.

1.5 Model-Driven Software Engineering

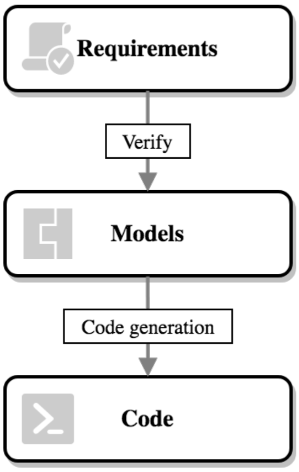


Figure 1.1: Model-Driven Software Engineering [Jürjens, 2005]

1.5.1 UML

Before exploring the rationale behind using the UML extension in the first place, it is essential in the first place to give a top-down overview of what UMLsec is. Since UMLsec is an extension of UML, utilizing the syntax and semantics of UML, we must first explore the benefits of using such an extension [Jürjens, 2005]. Using UML allows for many advantages. Firstly UML can be described as a de-facto standard in industrial modeling notations. This is because an increasingly large number of developers are trained in UML. The reason behind this is that most educational institutions teach modeling using UML, and also there exist a large quantity of educational resources on the approach. Consequently more proficient users are obtained.

Another main advantage of UML, is that it provides graphical intuitive description techniques with multiple views of a system, through various diagrams (classes, sequences, actions etc.). With UML it is possible to model a system on a high level and all the way down to the smaller parts/modules/components on a lower level.

In terms of details, UML is relatively precisely defined, since it defines syntax and semantics of the UML notation in a relatively high degree of detail. Although the syntax and semantics of UML might be well defined, the notation is arguably not entirely formal and the completeness of models is hard to determine. These are some of its drawbacks. This means that the semantics and syntax of UML can – for a given model of a system/module/class – allow many different syntactical and semantically correct models. This can be disadvantage in terms of the possibility of producing models of a wide range in terms of quality.

One of the benefits of UML being a de-facto standard is that there is good tool support. This further fuels its role as a standard, as most educational institutes and developers prefer to use UML for this exact reason. Therefore there is a low threshold for using UML. Furthermore when modeling in UML, a variety of tools exist that provide the basic functionality required to use the various diagrams.

1.5.2 ThingML

ThingML[2] is Domain Specific Modeling Language (DMSL) and code generation framework that allows developers use models to implement a system, and the models are used to generate code. The language describes software components and communication protocols, where the formalism consists of a combination of architecture models, state machines and an imperative language. This is imperative in the sense that it uses statements to change the state of the program. The toolset for ThingML consists of text editors for the purpose of creating and modifying the models, a set of transformations to create diagrams from the models and lastly a set of code generators to compile the language to code (i.e. Java, C and Javascript).

To best understand what ThingML is and why it is a notable approach to developing applications for IoT, we must first look at the objective of MDSE (Model Driven Software Engineering). The general concept of MDSE is to start by producing a model of the system, from a set of requirements and finally implement a system from the model as shown in figure 1.1. The implementation can be done in two ways, either automatically (using e.g. ThingML) or manually.

The general sense behind MDSE is the effectiveness and ease of automatically generating code from models. Although it has its issues, there are many areas where this method of developing software (SW) has proven to be advantageous. Many advocates of MDSE mention *"increase of productivity"* as the main factor for developing in this manner, but there has been observed some drawbacks too. [Harrand et al., 2016] lists one of the biggest drawbacks of MDSE connected to code generators being in the form of "black-boxes" that are not easy to trust with the quality of code. The other drawback of MDSE is that the models and their respective languages are not necessarily suited for code generation.

ThingML, on the other hand has been developed to address the drawbacks associated with code generating in terms of MDSE. ThingML is not only a modeling language but also a code generation framework tailored for the heterogeneous and complex nature of modern systems, in the sense that it is a highly customizable and supports multi-platforms. ThingML also

[2] `http://thingml.org/pmwiki.php?n=Main.HomePage`

addresses some concerns of MDSE by providing customizability to its code, and an abstraction that the developers require to be able to improve productivity. In short, the main idea behind ThingML is to enable the ability to produce source code, in a more efficient way, something that was not easily guaranteed by MDSE.

1.6 Model-Based Security Engineering

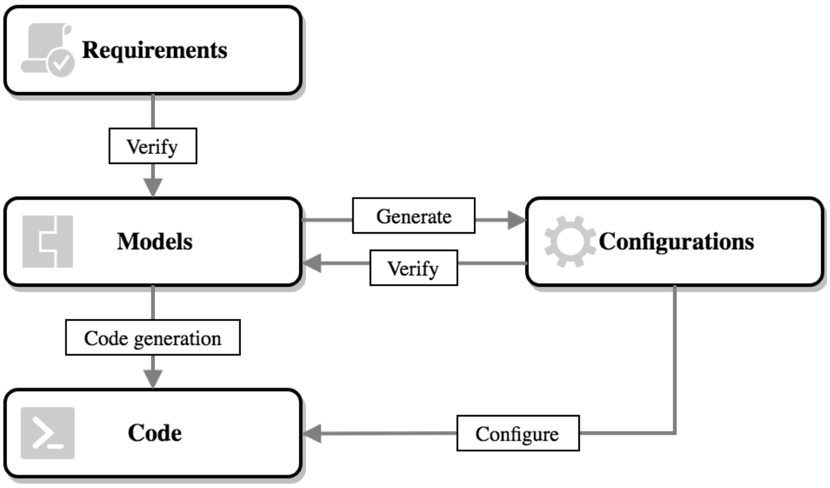


Figure 1.1: Model-Based Security Engineering [Best et al., 2007]

Since the main focus of this thesis is based on model-driven development using the ThingML framework, we need to address model-driven development in terms of security. The main role of MBSE (Model-based Security Engineering) is to provide a solid approach for developing secure systems. Firstly the initial steps of model-based development still apply as shown in figure 1.1. However in terms of MBSE, we first need to analyze the model to check if the system fulfills the security requirements on the design level. Thereafter we generate the test sequences from the model, to check if the code is in fact secure. Furthermore with this approach, the security requirements and assumptions of the system environment can either be specified within the Unified Modeling Language (UML) extension specification UMLsec or within the source code. The method in discussion for this thesis is using UMLsec.

1.6.1 UMLsec

To fully understand the importance of UMLsec, we must look at the different benefits and drawbacks for using this extension. We have established the motivator for using UML within the context of Model-Based Security Engineering (MBSE). However in terms of model-based security engineering, we need to investigate how UMLsec can fit in this context.

There are many reasons why the UMLsec extension is suitable for model-based security engineering. One of the main incentives for using UMLsec is that it allows one to evaluate UML specifications for security weaknesses on the design level. Within the development process, the design process is one of the earlier activities. By evaluating for security weaknesses early in the development process, we can discover logical holes and potential errors, and furthermore be able to solve early flaws before they become time and cost-consuming.

The second incentive for using UMLsec is that it encapsulates established rules for prudent security engineering in the context of a widely known notation. Using a well know notation and de-facto standard as the basis and context can be a major advantage in terms of communicating and understanding the predefined security specifications.

Another advantage of using UMLsec is that it allows the developer to consider security requirements from early in the system development process. This is arguably one of the best ways of developing secure systems is in-cooperating security as early as possible. Since one of the main issues today is the trade-off between developing highly secure systems and developing cost-effective systems, one way of decrease this trade-off is early security engineering.

Another incentive for using UMLsec is that it includes little additional overhead since the UML diagrams can serve as system documentation. This is further strengthened by the fact that UML is a de-facto standard, where a high number of developers are trained in UML.

One must also emphasize that UMLsec also have its weaknesses. Since UMLsec is an extension of UML, it is important to mention how the drawbacks of UML because they also affect its extension, UMLsec. Arguably the main weakness of UML is tied to its formality. Although UML is very precisely defined and provides a high degree of detail, it is nevertheless not entirely formal.

1.7 Related Approaches

The following approaches are related and alternatives to UMLsec. They are derived from a survey by [Talhi et al., 2009], where he weighs the different approaches against each evaluate the usability. We shall briefly look at these approaches and frameworks to discuss what separates them from UMLsec, and finally provide an overview of how they specify security requirements.

1.7.1 Alternatives

Aspect-oriented Approach

The Aspect-oriented approach proposes the modeling of access control policies, and only limited to these types of policies. This is done by supplementing UML with new diagrams that represent the schemes of (1) Role-Based Access Control (RBAC), (2) Mandatory Access Control (MAC) and (3) Discretionary Access Control (DAC).

The MAC, MAC and RBAC are separated from the main design, thus making it easy to alter the design without impacting the entire security of the application. These schemes are further decomposed into security feature representing specific elements of access control policies (e.g. permissions, delegation rules etc.)

Zisman's Framework

The Zisman's framework is a approach that aims to support the design and verification for secure peer-to-per applications. The framework uses UMLsec for designing models and specifying security requirements. In addition to representing possible attack scenarios and potential threats, the framework models abusive cases (i.e. behavior) thus helping designers to identify the security properties to be verified in the system. The framework also defines a graphical template language for expressing properties to be defined.

SecureUML

SecureML is an approach to model RBAC policies for model-driven systems, by providing a general schema for systems development through combination of design modeling languages with security modeling language. Therefore it does not fix one specific design modeling language. A delimiting feature of this framework is that it only focuses on specifying RBAC model, and also does not support secure code generation.

SOCLe

The SOCLe project provides a framework that integrates security into software design. It also provides the verification of UML specifications and graphical user interface tool that allows the visualization of the verification result. Using this approach and framework, the security policies are simply specified using Object Constraint Language (OCL) (see Chapter 5).

For-LySa

The ForLySa approach proposes two UML profiles to model authentication protocols, which are the following:

(i.) *Static For-LySa profile*: describes how the authentication protocol concepts (i.e. Server, Keys, Messages etc.) can be modeled using UML diagrams.

(ii.) *For-LySa profile*: models the dynamic aspects of the protocol in sequence diagrams, including the information needed to analyze the protocol.

For validation of the protocol, the approach defines a specification language with semantics to write pre/post-conditions and constraints. A delimiting feature with this approach is that it exclusively focuses on modeling authentication protocols.

AuthUML

The AuthML framework focuses on incorporating access control policies solely into use case diagrams. The framework aims toward analyzing over modeling (althrough not excluded) of access control policies during the early stages of the software development process before proceeding to the design modeling. This is to ensure that the requirements are complete, consistent and conflict-free.

Summary

[Talhi et al., 2009] distinguishes the various approaches for specifying and enforcing security requirements at the design level with UML based on three approaches:

i. Approaches based on artifacts by UML specification.

ii. Approaches that require extending the UML metalanguage.

iii. Approaches that require explicit extension of the UML metalanguage.

The first kind of approach uses the three following UML artifacts for security specification (i.) Stereotypes, (ii.) Object Constraint Language (OCL) and (iii.) behavior diagrams. The second kind of approach specifies by either (i.) extending the UML metalanguage or (ii.) creating a new metalanguage. To distinguish which contribution utilizes which approach, [Talhi et al., 2009] also summarizes the comparison of the approaches using the following table 1.1. The ❝✓ denotes the contributions we have made while adopting and extending UMLsec together with the Papyrus tool. In other words the tool helped us enforce security requirements using more approaches than originally mentioned by the source.

Table 1.1: Comparison of the related approaches

	Stereotypes & tags	OCL	Behavior diagrams	Extend UML metalanguage	New UML metalanguage
UMLsec	✓	❝✓	❝✓	❝✓	–
Aspect-oriented	✓	–	–	✓	–
SecureUML	✓	✓	–	–	✓
Zisman	–	–	✓	–	–
SOCLe	–	✓	–	–	–
For-LySa	✓	–	–	–	–
AuthUML	–	✓	–	–	–

❝✓= Our contribution to this work.

The report conducts a usability survey to compare the various approaches and weighs the different advantages and drawbacks for each contribution. Finally, the report concludes that efforts for verifying the design against security requirements are very important, regardless of the chosen approach. This implies that the tool you use for the job is not so important, just as long as the job itself is being done.

1.8 Outline

This thesis is divided into five parts (see Fig. 1.1) as following:

P1: Introduction: introduces the main topic and relevant concepts, and also describing the context and use case of our topic. *i.e. Chapter 1.*

P2: Background: sets up the in-depth background knowledge, fundamental for the remaining parts of the thesis. *i.e. Chapters 2 and 3.*

P3: Design: shows the process of designing for UMLsec through defining and constructing a profile, and specifically the work behind the profile extension. *i.e. Chapters 4 and 5.*

P4: Application: shows the process of applying and using the UMLsec profile to the actual use case of ALCCS. *i.e. Chapter 6.*

P5: Discussion: focuses on discussing the outcome related to the use of UMLsec, the actual advantages and/or limitations of this approach, the future of this approach in terms of use in other projects etc. *i.e. Chapters 7 and 8.*

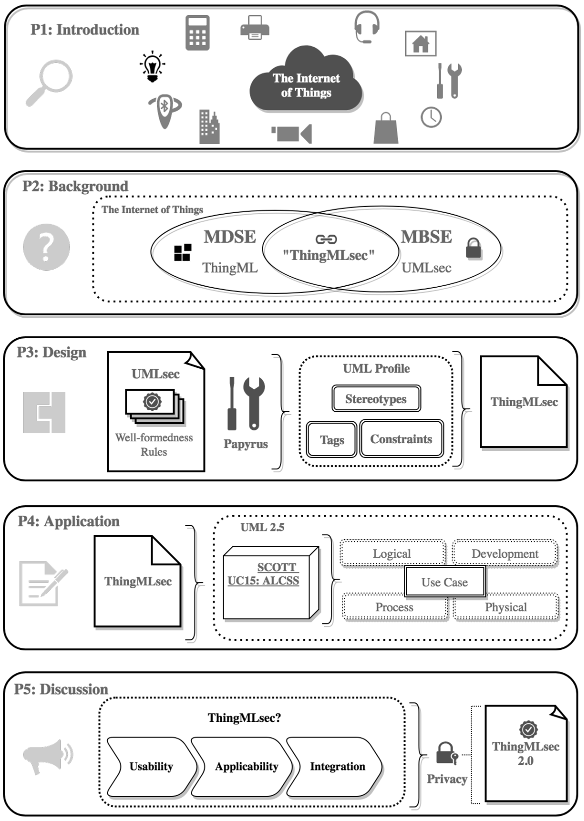


Figure 1.1: Structure and Outline of the Thesis

Furthermore the parts are divided into the following chapters:

- *Chapter 1: Introduction*
  Introduces the motivation, theme, problem and relevant approaches regarding the thesis.
- *Chapter 2: ThingML*
  Introduces the MDSE approach of ThingML; a code generation framework.
- *Chapter 3: UMLsec*
  Introduces the MBSE approach of UMLsec for secure systems development.
- *Chapter 4: UML Profile Extension*
  Provide the information (i.e. rules) for creating a UMLsec profile.
- *Chapter 5: UMLsec Profile Construction*
  Discuss the available and needed tools for UML profiling, and finally create our own extension based on UMLsec – namely ThingMLsec.
- *Chapter 6: The Application of ThingMLsec to ALCCS*
  Introduce the case and relevant use case scenario, and thereafter apply and verify the stereotyped ALCCS models with security requirements and their constraints.

- *Chapter 7: Conclusion*
  Provide a summary, discuss and measure the process of our using our approach to handling security.
- *Chapter 8: Future Work*
  Possible ways of proceeding, to further build and enrich our work.

Part II

Background

CHAPTER 2

ThingML Code Generation Framework

We have previously established that the main goal behind the conception of Model-Driven Software Engineering (MDSE), was to increase productivity and decreasing high costs of developing software, by generating code from models. However the pre-existing trade-off between the increase of productivity and the reduction of cost, makes this difficult. The main issue with MDSE approaches is firstly that modeling languages are being used for other purposes not suited for code generation. The other issue is that code generation frameworks work as "black boxes", where developers have little insight and control on the code being generated. This makes it difficult to trust and may result in suboptimal solutions.

Since ThingML was built with the intension to minimize the main drawbacks of MDSE, this section will give an overview and insight in the code generation framework that characterizes ThingML, and attempts to uncover how it's built and operates. The main idea is to shed light on the key structures and parts that it relies on – *Configurations* and *Things*, and how they work together to form ThingML. This section will provide further details on the so-called *extension points* within the key structures; i.e. what role they play and why they exist.

ThingML – being a modeling language and tool designed for code generation – aims to become far more customizable in order to address the main issues of MDSE. Therefore ThingML provides more customizability of it's code, including giving developers the abstraction that they need to improve productivity to reduce the idea of a "black-box" with increased insight in the code generation process. Thereby making it a *"highly customizable multi-platform code generation framework"* in short [Harrand et al., 2016].

2.1 Structure of ThingML

To further understand whats make ThingML unique among the MDSE frameworks, we need to look at *how* it is made up. Therefore in terms of how ThingML as a tool and framework

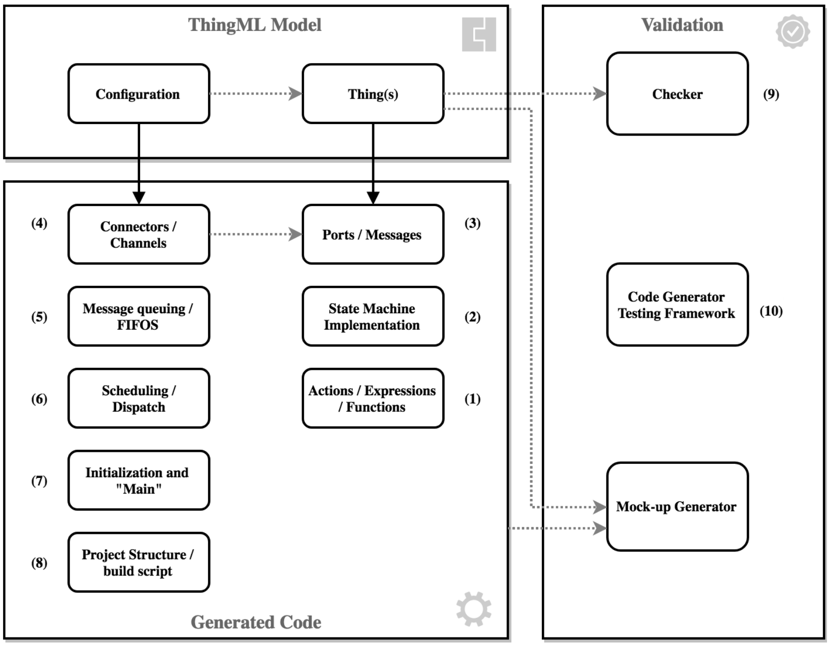


Figure 2.1: Overall structure of the ThingML DSML [Harrand et al., 2016]

is constructed, we can look at the two parts that make up for its structure. Firstly you have the *Thing(s)*. The "thing" represent the software components. It is also the implementation unit/component/process, depending on use, context and/or approach. Then there is the *Configurations*, that describes the representation of the software components (i.e. "Things").The configurations can also be described as a set of instances of the defined "things" and the set of connectors between two parts of an instance. On a lower level, we have *extension points* that exist within the two main structures.

Extension points are essentially abstract classes or interfaces in the code generation of ThingML, with a set of methods responsible for generating code for their respective model elements (i.e. Configuration or Thing(s)). There exist 10 extension points in the ThingML, that are separated between the two model elements.

2.1.1 Thing(s)

A "Thing" can be shortly defined as the representation of a software component. This can define variety of different elements, like properties, functions, messages, ports and a set of state machines [Harrand et al., 2016]. These properties are stored locally within a Thing, as *variables*, where they can be accessed using functions. However these variables are not accessible by other things. A Thing's only public interface within a thing is through ports, which can receive and send a set of messages.

The first 3 (three) extension points are directly related to the code generation of "Things". These are depicted on fig. 2.1.

1. Actions/Expressions/Functions
The first extension point within the framework, is related to the code generation that corresponds to code for action, expressions and functions in a Thing.

2. Behavior implementation
The second extension point corresponds to the code generation from the state machine and Event-Condition-Action (ECA) contained in things.

3. Ports/Messages/Thing APIs
The third and final extension point of the Thing element, corresponds to the wrapping of "things" into reusable components on the target platform.

2.1.2 Configurations

Configurations essentially describe the thing(s) interconnection [Harrand et al., 2016]. Within this part of the ThingML model, there are five (5) extension points that are related to the code generation.

The second group of extension points corresponds to the Configurations (see fig. 2.1). Some of the extension points are not directly connected to the generation of code, but supports the code generation by providing validation, packaging and generally setting up the code.

4. Connectors/Channels
This extension point is responsible for the code generation corresponding to the connectors and transporting messages from one thing to the next.

5. Message queuing/FIFOs
This extension point is used to tailor how messages are handled when the connectors are linking two things running on the same platform.

6. Scheduling/Dispatch
This extension point is in charge of generating code which orchestrates the set of things running the same platform.

7. Initialization and "Main"
In this extension point the objective is to generate code for the entry point and initialization

code in order to set up and start the generated application on the target platform .

8. Project structure/Build script

This is not an extension point directly related to generating code, but it is responsible for providing the required file structure and build script in order to make the generated code well packaged and easy to compile and deploy to the target platform.

The last two extension points are related to the validation of code, within the ThingML framework. In figure 2.1, these two extensions are visualized and modeled as part of the validation part of the framework.

9. Checker

This is the part of the framework that provides support for validating an input ThingML Model before generating code from it. Furthermore this part includes the syntactical checks and a customizable set of rules that apply to the application logic expressed in a module. The rules are customizable, making it possible to extend the set of rules, and to enrich the preexisting compilers.

10. Code generator testing framework

The tenth and final extension point has the purpose to validate code generators, and currently consists of 140 extensible tests.

CHAPTER 3

UMLsec

3.1 Security Engineering

During Chapter 1, we discussed how the increasing interconnection is adding new potential vulnerabilities. The increasing impact of such attacks makes this a pressing matter, and therefore this section will go more into the actual security measures that can be added to increase the level of security in such systems using MBSE.

In terms of Security Engineering, [Jürjens, 2005] mentions two issues that need to be addressed when attempting to develop secure systems. One of the issues today is that systems communicating over various links are prone to many different attacks. This is especially regarding communication over open, unprotected networks. In terms of attacks over the communication links, an adversary is capable of reading, inserting and or/deleting messages exchanged over the communication links. Some ways to prevent or minimize this threat, is *encryption* and *cryptographic protocols*. These are some of the most commonly known methods to ensure secure communication.

The other issue that needs to be addressed in terms of developing secure systems, is direct attacks on the physical nodes. The main concern is that getting access to data and manipulating the data can be fairly easy, especially if there are discrepancies in protection of the physical or hardware component(s) of the system. There are many different ways of implementing security measures. A probable scenario is if an attacker acquires access to one of the aforementioned physical nodes, a possible security measure can be to use protected hardware measures like "smart cards". In certain systems this might be a necessary security measure to treat the potential vulnerability.

A unique feature of UMLsec is that it is designed for more Object-Oriented (OO) and component based systems [Jürjens, 2005]. This approach adds security in general by adding two very important elements. The first is *method calls.* With methods calls there is an added general mechanism for controlling access to data. This also adds the possibility of validating every access or request made to a certain subset of the system. The other element is *information hiding.* This mechanism is achieved by the encapsulation of data in objects. The result of this is that data within an object, can only be accessed within objects, and messages are the only way to communicate.

3.2 UML Extension Mechanisms

To go more specifically into the characteristics of UMLsec, we can start by addressing the nature of UMLsec. UML offers three (3) main "lightweight" language extension mechanisms, which are *stereotypes*, *tags* and *constraints.* These mechanisms are expressed in a *profile.* In short, stereotypes are used together with tags to formulate the security requirements and assumptions.

3.2.1 UMLsec Stereotypes

The central idea with UMLsec, is to use *stereotypes* to define new types of modeling elements that extend the semantics of existing types or classes in the UML metamodel. This can be done on the physical og logical level of the system or eventually as policies for the system to obey.

Security Assumptions

The first type are the *security assumptions*, and they add security-relevant information on the *physical* level of the system. This is for example the stereotypes of `«Internet»`, `«LAN»` or `«wire»`.

Security Requirements

The other type of stereotypes is *security requirements.* These add security relevant information on the *logical* structure of the system or on specific data values. This is for example the stereotypes of `«secrecy»` for the system, or `«critical»` for specific data values.

Security Policies

Security policies summarizes the protection requirements of a system, and the policies that system parts are supposed to obey. There are several such policies in UMLsec like `«fair exchange»`, `«data security»`. The most important policies are as follows:

1. ***Fair exchange***

The policy of *fair exchange* assures that the trade (in terms of trading good) is performed fairly, preventing participating parties from cheating one another. The general idea is relatively clear from the name of the policy – that the exchange should be performed fairly in regards to the parties.

2. Non-repudiation

With *non-repudiation*, the idea is that the action in discussion cannot be successfully denied. This, in the sense that it is "provable", usually with respect to some trusted third party. The *non-repudiation* security policy can be used to enforce the *fair exchange* policy. If one can assure the fairness of an exchange by including an unbiased and trusted third party, that further improves fairness and prevents parties from cheating one another.

3. Role-based access control

Role-based access control (rbac) is related to how access control managed within a system. To control access to protected resources, the idea is to keep permissions manageable by not assigning them to users, but *roles* instead. Each user's permissions and level of access control is based on what type of role they have within the entirety of the system.

4. Secure communication link

To preserve *secrecy* and *integrity* of data during transmission, sensitive communication links between ports/modules/components need to be protected. This is especially a very important policy because of the growing nature of interconnected systems.

5. Secrecy and integrity

A vital requirement in terms of data security is *confidentiality* (secrecy) and *integrity.* With *confidentiality*, the idea is that data is read only by legitimate parties. On the other side with *integrity*, the idea should be modified only by legitimate parties.

6. Freshness

The idea behind the *freshness* policy is that messages created and/or sent during the current execution of the system, should not allow replays of older messages by the user or adversary. In regards to this policy, it is usual to use a so-called "*nonse*", which is a random value to be used once, to enforce the freshness.

7. Authenticity

This security policy is divided into two separate types of *authenticity.* At first, you have the *message authenticity.* This type of authenticity is about being able to trace back the piece of data to the original source (i.e. origin). The second type of authenticity is *entity* authenticity. This type is about being able to identify a participant, and to actively confirm the level of participation within a given time.

8. Secure information flow
Before diving into the policy, we can identify many systems as having a multi-level security structure. These levels are often different levels of data sensitivity. In general there are mainly two levels; high and low. *High security level* is the highly sensitive data (e.g. personal/security number, account number etc.), and the *low security level* is the least sensitive data (e.g. number of saved items, nickname etc.). In such multi-leveled systems, different trusted parts may interact with untrusted parts. With this type of interaction, one has to ensure that there is no leakage of (sensitive) information.

To ensure the protection of sensitive information in multi-level security systems, there are therefore two policies; "no down-flow" and "no up-flow". Firstly the "no down-flow" means that the high security level data (trusted data) may not influence the low security level data, but the opposite is permitted. On the other side, the "no up-flow" means that the low security level data (untrusted data) may not influence the high level data, but the opposite is permitted. These are security properties that are called secure information flow.

9. Guarded access
The security mechanism that is *access control*, ensure that only legitimate parties have access to a security relevant part of the system. In this security policy, we are talking about access control being enforced by *guards*.

3.3 Security Analysis & Adversary Model

In this section, we will briefly explain an underlying aspect of UMLsec that will be used throughout the thesis, and specifically as part of profile extension and to further provide security relevant information with the use of the stereotypes, tags and constraints.

An essential part of analyzing security is being able to address the potential adversaries internally and/or externally capable of harming our system. With UMLsec we can model specific types of adversaries that can attack different parts of our system. An example of this is that we can have a attacker of the type *insider*, that is capable of harm through the communication links in a company wide Local Area Network (LAN).

So how does UMLsec model the potential adversaries that can attack our system? In terms of security analysis of the subsystem specification, this is done by modeling the adversary behavior. The first step is to define the set of *abstract threats* {`delete`, `read`, `insert`, `access`} respective to the specification of the physical layer of the system. The idea is that we should be able to model specific types of adversaries that can attack various parts of the system in a specified manner. The idea is that the adversary actions `read`, `delete` and `insert`, meaning that the adversary may read, delete and insert messages on a communication link. For nodes, the threat of `access` means that the adversary may directly access the physical node in the system.

Furthermore UMLsec assumes a function $Threats_A(s)$ which takes an *adversary type* A and a stereotype s and returns a subset of the *adversary actions* of {`delete`, `read`, `insert`, `access`} [Jürjens, 2005]. The general idea with these functions is that they arise from the specification of the physical layer of the system, particularly using deployment diagrams. This is explained in further detail in chapter 4. Henceforth the objective is to using the function $Threats_A(s)$ to specify the *threat scenario* associated with the adversary type A, against either a component or communication link stereotyped s.

With this in mind, we can state that the *threat scenario* determines two things. The first is which *data* the adversary can obtain by accessing the components (i.e. nodes). The second thing is which *actions* the adversary is permitted by the threat scenario to apply to the concerned links (i.e. communication links). From this, and more specifically the *abstract threats* we can derive the more basic *concrete threats* used in the modeling and analyzing the possible adversary behavior.

The modeling and analyzing of possible adversary behavior is done in the following way. First we have to require that the adversary needs to *access* the system part for it to be attacked. In theory this means that for example that an attack can only happen on a wireless network, if the adversary is within physical proximity. The second step of modeling and analyzing adversary behavior is broken down to the to the atomic actions of `delete`, `read` and `insert`, with relation to the communication links connected to the nodes. Thereby if the adversary access the node contained in the system, this means he may *delete*, *read* and *insert* messages on the communication links.

Given a UML subsystem S the function of $Threats_A(s)$, consisting of a link or node x in a deployment digram gives rise to the function $threats_A^S(x)$ of concrete threats from adversaries A and returns a set of strings $threats_A^S(x) \subseteq \{delete, read, insert, access\}$. This allows for evaluation of UML subsystems through their execution semantics by referring to the security framework using UML Machine Systems. The latter is not covered in this thesis, but is described in detail in [Jürjens, 2005].

UMLsec provides example of threat sets associated with some common adversary types and are depicted in Fig. 3.3 and 3.3.

Fig. 3.3 describes an example of the default (i.e. average) attacker with the modest of capabilities, meaning that there naturally is a possibility that a more sophisticated and capable adversary might do more harm than the aforementioned *default* attacker, even on the more restricted communication links (e.g. `«wire»`). What is assumed by the *default* adversary's capabilities is that this type of attacker is able to *read*, *delete* and *insert* messages that are on the `«Internet»` communication link. On the `«encrypted»` Internet link however, like Virtual Private Network (VPN), the attacker would not be able to read, or insert messages properly encrypted (i.e. assuming attacker does not easily gain access the encryption key). However the attacker might still be able to delete messages regardless of being encrypted or not (e.g. bringing down the network server). Moreover the *default* attacker is assumed not to have access to either a Local Area Network (LAN) or `«wire»`. Furthermore the attacker is assumed not to have access to or eavesdrop on connections or security-critical devices like smart card, Point-of-Sale (POS) devices or issuer node. It is well noticed that a more sophisticated attacker might be able to gain access to some of these devices and connections, but the main concern here is the default attacker.

Stereotype	$Threats_{default}()$
«Internet»	{delete, read, insert}
«encrypted»	{delete}
«LAN»	$\emptyset$
«wire»	$\emptyset$
«smart card»	$\emptyset$
«POS device»	$\emptyset$
«issuer node»	$\emptyset$

Table 3.1: Threats from the *default* attacker

To showcase a different type of attacker, Fig. 3.3 shows the *insider* attacker and his/her capabilities. This is merely an example of a *concrete* attacker provided by [Jürjens, 2005]. The main idea is that as an insider, the adversary might access e.g. the encrypted Internet communication link and access the local system components, given that he/she might know the corresponding keys (i.e. to decrypt).

Stereotype	$Threats_{insider}()$
«Internet»	{delete, read, insert}
«encrypted»	{delete, read, insert}
«LAN»	{delete, read, insert}
«wire»	{delete, read, insert}
«smart card»	$\emptyset$
«POS device»	$\emptyset$
«issuer node»	{access}

Table 3.2: Threats from the *insider* attacker

Nevertheless, for security in IoT systems, we need to address that the communication occurs over other links that are not addressed by [Jürjens, 2005] and the *default* attacker model in Fig. 3.3. During Chapter 1, we explained the communication of IoT enabled by *5G networks*, and the newest update of *Bluetooth* – Bluetooth Low Energy (BLE). During the following subsections we discuss the threats associated with these types of communication links, and how they can be addressed. Finally our findings will provide a more domain-specific threat scenario for the attackers within IoT and therefore a more updated *defaultIoT* attacker.

3.3.1 5G Security

As previously mentioned during the introduction, we defined 5G cellular networks being distinguished by two types of two-tier networks – microcell tier (BS-to-device communication) and device tier (D2D communication). In this thesis we are more concerned with the Device-to-Device (D2D) communication or device tier. The directness of the device tier communication provides many security and privacy concerns over a potential attacker mainly because of the lack of a operator control in the form of a base station. Since user data is transmitted and routed between various devices, this requires security (and privacy) to be maintained. A possible solution for this is to close access for the devices that want ot operate in the device tier [Tehrani et al., 2014]. More specifically this means that a device has a list of "trusted" devices and the excluded devices (and potentially malicious devices) must use the microcell tier (BS-to-device) to communicate, where there is added security. There is also a possibility for the "trusted" devices to set up a proper encryption amongst each other to avoid threats like eavesdropping, Man-in-the-Middle (MITM) attacks etc. The technical details of how one can either encrypt devices within a device tier communication, and setup of a closed access D2D will not be covered by this thesis, but an example of this can be found in [Yue et al., 2013]. This report proposes the "Secrecy-based Access Control for D2D communications for cellular networks (5G)", and is a possible option for addressing security issues with device tier communication.

3.3.2 BLE Security

We mentioned that a form of D2D communication within IoT is through Bluetooth, and specifically the most current version labeled BLE. The default attacker is perfectly capable of targeting the wireless connection between devices, so therefore a requirement for D2D communication is to protect the links through a form of encryption. Bluetooth, like many D2D communication technologies is sustainable to several wireless networking threats and vulnerabilities like eavesdropping, MITM, Denial of Service (DoS) attacks etc. So to improve the security for Bluetooth and BLE, it is highly recommended to use the strongest Bluetooth security mode available.

There exist specific security modes for specific Bluetooth version. However for our use case and for this thesis we are mostly concerned with Bluetooth Low Energy (BLE) (i.e. introduced in Bluetooth version 4.0 and updated in 4.1, 4.2). BLE is recommended to use the Security Mode 1, Level 4 [Padgette et al., 2017], and this is the strongest mode because it requires authenticated low energy secure connections pairing with Elliptic Curve Diffie Hellman (ECDH) based encryption. To briefly describe the other security modes and levels, we have Security Mode 1, Level 3 that requires authenticated pairing and encryption but does not require the ECDH-based cryptography and this limits the protection from eavesdropping. Other security modes/levels allow unauthenticated pairing, which means no protection from e.g. MITM attacks etc. provided during the cryptographic key establishment. These modes vary depending on the Bluetooth specification version for the related device, so it is highly recommended to use the most secure security mode available.

To give an overview of the security modes and levels available for BLE, we can look at Tables 3.3 and 3.4. Security Mode 1 visualizes the multiple levels that are associated with encryption, while the Security Mode 2, shows the multiple levels associated with data signing.

Level	Security Requirement
L1	Specify no security (i.e. no authentication and encryption)
L2	Requires unauthenticated pairing with encryption
L3	Requires authenticated pairing with encryption
L4	Requires authenticated communications with data signing (BLE ver. 4.2)

Table 3.3: Encryption-related levels of BLE Security Mode 1

Level	Security Requirement
L1	Requires unauthenticated pairing with data signing
L2	Requires authenticated pairing with data signing

Table 3.4: Data-signing-related levels of BLE Security Mode 2

3.3.3 Conclusion

We have therefore established that for 5G, the device tier communication in its raw form is not very secure, but that there are configurations and measures one can account for to fix and improve this. With this in mind we can enclose that by adopting these security measures, either encryption, closed access and/or secrecy based access control for 5G networks we can achieve a `«secure 5G»`, for which in our threat scenario is more safeguarded and a better option to pure 5G cellular communication, at least in terms of adversary capabilities.

For Bluetooth and specifically BLE, we have established that the best way to provide secure D2D communication is through the use of the most recommended security mode and level, and for BLE ver. 4.2 this is the Security Mode 1, Level 4 that provides the best security. However for the other versions of BLE Security Mode 1, Level 3 is sufficient, but will not guarantee the best security. This may however be the highest available security mode and level for certain versions of BLE. For the use case for the demonstration and for our threat scenario we merely enclose the highest and recommended security mode and level generically as `«secure BLE»`. In other words, when we refer to *"secure BLE"*, for BLE ver. 4.2 this is Security Mode 1, Level 4, and for other BLE ver. <4.2, this is the Security Mode 1, Level 3.

Therefore we have adopted the specified adversary abilities for the *default* attacker depicted in Fig. 3.3, and from this added communication links and analyzed potential adversary capabilities for specifically IoT systems and devices to define a domain-specific *defaultIoT* adversary. This new adversary scenario is the one we will use to validate and enforce security requirements for in Chapter 6.

Stereotype	$Threats_{defaultIoT}()$
«Internet»	{delete, read, insert}
«5G»	{delete, read, insert}
«BLE»	{delete, read, insert}
«secure 5G»	{delete}
«secure BLE»	{delete}
«encrypted»	{delete}
«LAN»	$\emptyset$
«wire»	$\emptyset$

Table 3.5: Threats from the *defaultIoT* attacker for ThingMLsec

Part III

Design

CHAPTER 4

UML Profile Extension

We have established in Chapter 3 that UML provides *extension mechanisms* to allow modelers to extend the modeling language without actually changing the underlying modeling language, and that there are several types of such stereotypes. The main idea is to use these extension mechanisms to further define new concepts. In this section, the goal is to group these extensions into a *profile*. [Rumbaugh et al., 2010] defines a profile as a "*coherent set of extensions applicable to a given domain or purpose*". We also mentioned the extensions mechanisms; *stereotypes*, *tags* and *constraints* that are used by UMLsec to extend the UML.

The fundamental premise of profiles is that the domain-specific concepts are derived as extensions of existing UML concepts. Therefore the goal with this chapter is also to try and determine exactly which of the UML concepts are to be extended through extensions and with what type of information.

4.1 Background

Since we don not want to "reinvent the wheel", we merely adopt UML and tailor it to our needs – which are to enforce security requirements at the designing and specification level with UMLsec. The way to do this is through the profile mechanism of UML, which is the focus for this part of the thesis.

4.1.1 UML Profiling

The current evolving complexity of systems require new and better design paradigms and approaches. One way of doing this is by separation of concern in the field of model-based engineering, where the idea is specialization of technologies has led to the development Domain Specific Modeling Language (DMSL) [Gérard, 2011]. The idea is that the DMSLs (i.e. UMLsec) provide certain constructs that are relative to the concepts of the domain in question (i.e. security). Currently many systems are being modeled using the general purpose languages, like UML. The main reason behind this is the availability of tools, documentation of the language and training material for the languages, and finally the availability of experts and expertise. For others, the use of general purpose languages comes down to the limited risks and costs connected to them. Most domain-specific program libraries are developed and used, written in general purpose languages. However the downside of this method is that the tools are not domain-oriented, leading to external tools – sometimes automated – like compilers, not being able to recognize and take advantage of the domain-specific knowledge. An example of this is when domain-specific libraries lack compilers or other testing tools, then the benefit and use of this approach is therefore limited, and eventually the usefulness of using model-based engineering for designing complex systems is limited.

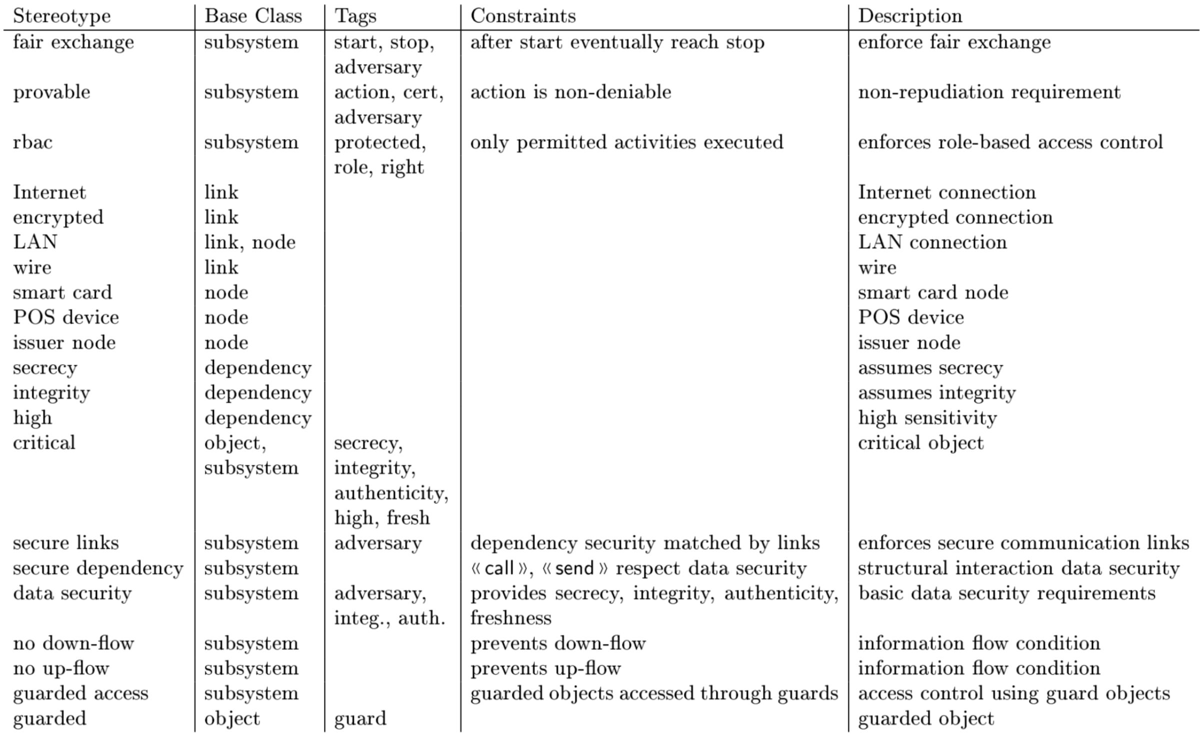
One important advantage of using UML profiling is that the language can be defined in such a manner that it is suitable to a specific problem. However UML specifically, provides the concept of *profile* allowing us to derive domain specific modeling languages for from its set ot general purpose language concepts. The main and most important advantage of this is that it allows us to reuse existing UML modeling tools and naturally the expertise available for the modeling language. The other advantage of UML profiling, in comparison to using a different general purpose modeling language, is that the most of the domain-specific modeling languages for UML share a common semantic and syntactic foundation. For example, the concepts for package, composition, property etc and also the basis notations of object, class and interface. The common UML concepts are also common to many disciplines.

The profile is defined as a *package* identifying the subset of an existing base metamodel, and defining stereotypes and constraints that may be applied to the metamodel subset. An example of this is the stereotype of`«Internet»` extending the existing base metaclass of *"CommunicationPath"*. The intention is to make an extension to the UML metaclass and specifically to the model element of *"Communication Path"* to tailor it to a specific domain and application, which in this case is secure systems development using UMLsec.

4.2 Well-formedness Rules of UMLsec

We previously mentioned in chapter 3 that the UMLsec stereotypes can be grouped into various kinds of stereotypes; *security assumptions*, *security assumptions* and *security policies*. The idea is that the sterotypes define new types modeling elements extending existing classes in the UML metamodel. The *Secure Systems Development with UML* book [Jürjens, 2005] provides a complete list of the various stereotypes, tags and constraints available in UMLsec, in figure 4.1 (*stereotypes* and *constraints*) and figure 4.2 (*tags*).

Based on these definitions of the stereotypes in UMLsec depicted in the figure 4.1, we first need



Stereotype	Base Class	Tags	Constraints	Description
fair exchange	subsystem	start, stop, adversary	after start eventually reach stop	enforce fair exchange
provable	subsystem	action, cert, adversary	action is non-deniable	non-repudiation requirement
rbac	subsystem	protected, role, right	only permitted activities executed	enforces role-based access control
Internet	link			Internet connection
encrypted	link			encrypted connection
LAN	link, node			LAN connection
wire	link			wire
smart card	node			smart card node
POS device	node			POS device
issuer node	node			issuer node
secrecy	dependency			assumes secrecy
integrity	dependency			assumes integrity
high	dependency			high sensitivity
critical	object, subsystem	secrecy, integrity, authenticity, high, fresh		critical object
secure links	subsystem	adversary	dependency security matched by links	enforces secure communication links
secure dependency	subsystem		«call», «send» respect data security	structural interaction data security
data security	subsystem	adversary, integ., auth.	provides secrecy, integrity, authenticity, freshness	basic data security requirements
no down-flow	subsystem		prevents down-flow	information flow condition
no up-flow	subsystem		prevents up-flow	information flow condition
guarded access	subsystem		guarded objects accessed through guards	access control using guard objects
guarded	object	guard		guarded object

Figure 4.1: Stereotypes, tags and constraints in UMLsec [Jürjens, 2005, p. 51]

to understand the essential groups (i.e. Base Class, Tags and Constraints columns) and what they reveal about each stereotype. Because the definition of a stereotype must be consistent with the syntax and semantics of the UML metaclass it extends, we will look further into how the UMLsec defines its stereotypes and therefore the metaclasses they should extend during the process UML profiling.

4.2.1 Base Class

There are 5 (five) distinct base classes listed for each stereotype in UMLsec, and they are *subsystem*, *link*, *node*, *dependency* and *object*. To further understand what these base classes are and what purpose they have, we need to explain their role in the construction of a profile extension and specifically which metaclasses they should extend.

Subsystem

With a subsystem, we talk about grouping of model elements that represent a decomposed unit within a physical system. The list of stereotypes in Fig. 4.1, was listed when the current UML version was UML 1.5, and further defined subsystems as certain type of *packages*. Subsystems also integrate the different kinds of diagrams and between different parts of the specification of the system. It is worth mentioning that a subsystem modeling of the complete system, not merely a part, is called a *system* [Jürjens, 2005]. It is also possible to group parts of a model, that are represented by diagrams, into a package. In our profile extension based on the current version of UML 2.0, subsystems will therefore be identified as *packages* and extend the UML metaclasses of *Package* and *Model.*

Link

In UMLsec we mostly talk about links with regards to deployment diagrams, where nodes (displayed as boxes) are connected by solid lines representing communication links. Therefore within our profile extension, links will be the association between nodes that represents and extends the UML metaclass of *Communication paths.*

Node

The third base class is node. Similar to links, we talk about nodes in regards to deployment diagrams, where a node is defined as a run-time resource, like a computer or device etc. As previously mentioned, a node is displayed as a box, which can be connected by communication links. A node may also contain components that as displayed as rectangles, with two smaller rectangles on the left side.

Dependency

With dependency we talk about relationship between two model elements. [Rumbaugh et al., 2010] describes the various types of dependency relationships as; *realization*, *usage*, *trace*, *refinement*, and *binding.* Dependecy relationships can be between model elements in component, class or deployment diagrams.

Object

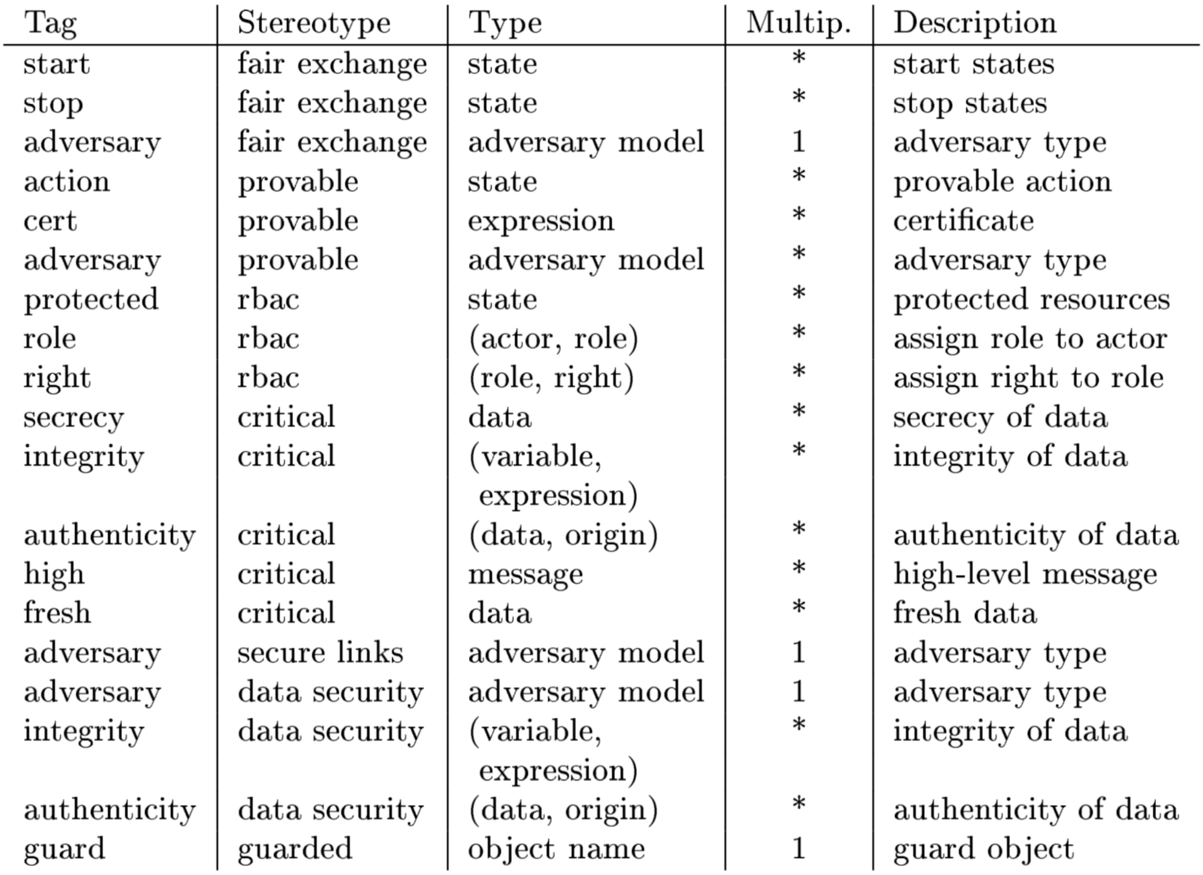
An object is "an instance of a class – that is, an individual with identity whose structure and behavior are described by the class" [Rumbaugh et al., 2010]. A class is a description of a set of objects, that share attributes, operations, methods etc. We can conclude that the base class of object therefore extends the UML metaclass of *Class.*

Summary

We have established that the 5 (five) base classes in the definition of stereotypes listed in figure 4.1. UML has many metaclasses, but the aforementioned five base classes represent the UML metaclasses that are to be extended by the stereotypes in the same figure. So now we can for example say that for the stereotype of «`fair exchange`» will extend the metaclass of *Package* or *Model*, since we have established that "subsystem" is a type of package, or model in UML 2.0.

4.2.2 Stereotypes, Tags & Constraints

The complete list of UMLsec tags in Fig. 4.2 lists 7 (seven) distinct stereotypes and their respective tags. To further understand the function of each of these tags, it is best to explain them from the perspective of their stereotypes, including all other related stereotypes.



Tag	Stereotype	Type	Multip.	Description
start	fair exchange	state	*	start states
stop	fair exchange	state	*	stop states
adversary	fair exchange	adversary model	1	adversary type
action	provable	state	*	provable action
cert	provable	expression	*	certificate
adversary	provable	adversary model	*	adversary type
protected	rbac	state	*	protected resources
role	rbac	(actor, role)	*	assign role to actor
right	rbac	(role, right)	*	assign right to role
secrecy	critical	data	*	secrecy of data
integrity	critical	(variable, expression)	*	integrity of data
authenticity	critical	(data, origin)	*	authenticity of data
high	critical	message	*	high-level message
fresh	critical	data	*	fresh data
adversary	secure links	adversary model	1	adversary type
adversary	data security	adversary model	1	adversary type
integrity	data security	(variable, expression)	*	integrity of data
authenticity	data security	(data, origin)	*	authenticity of data
guard	guarded	object name	1	guard object

Figure 4.2: Tags in UMLsec [Jürjens, 2005, p. 52]

`«fair exchange»` *(for use case diagrams)*

This stereotype applied to use case diagrams, represents the security requirement that any transaction should be performed in a way that prevents any two given parties, from cheating. However this stereotype, as opposed to the other stereotypes has an informal meaning. It mostly serves as an example of how we can use certain multiple stereotypes on multiple types of diagrams (e.g. use case diagrams and activity diagrams).

`«fair exchange»` *(for activity diagrams)*

With relation to acitivty diagrams, the `«fair exchange»` stereotype also assures that the trade is performed fairly. However in this case, we can use the tags {`start`}, {`stop`} and {adversary}. The {start} and {stop} have to at least specify the state (i.e. name). The {adversary} on the other hand, specifies an adversary type relative to the security requirement. The *constraint* for the associated stereotype and respective tags is that whenever a {start} state is reached within the activity diagram, then the {stop} state will eventually be reached, whenever the system is executed in the presence of an adversary specified the tag {adversary}.

Tags	**Type**	**Multipl.**	**Description**
{start}	state	*	start state(s)
{stop}	state	*	stop state(s)
{adversary}	adversary model	1	adversary type

Table 4.1: Associated tags for «fair exchange»

«secrecy», «integrity», «high»

These particular stereotypes are used to label dependencies of objects or subsystems in the static structure or component diagrams (i.e. class, deployment etc.). They represent the dependencies that are supposed to provide the respective security requirements (secrecy, integrity or protection of high-level messages) for the data sent along them either as arguments or return values for operations or signals. Moreover these stereotypes are used in the constraint for the stereotype «secure links».

«provable»

A subsystem (e.g. model or package) S may be labeled with the stereotype «provable» using the tags {action}, {cert} and {adversary}. The {cert} tag consists of an expression that acts as a proof that the action provided by the tag {action} was performed. The {adversary} also specifies an adversary type relative to the security requirement. The *constraint* of the stereotype therefore specifies that S may output the expression (i.e. proof) given in {cert} only after the state with the {action} is reached, whenever the system is executed in the presence of an adversary specified the tag {adversary}.

Tags	**Type**	**Multipl.**	**Description**
{action}	state	*	provable action
{cert}	expression E	*	certificate or proof of action
{adversary}	adversary model	1	adversary type

Table 4.2: Associated tags for «provable»

«rbac» *(for activity diagrams)*

To enforce the security policy of *role based access control*, we may use the corresponding stereotype «rbac» and the associated tags {protected}, {role} and {right}. The {protected} holds the state(s) within the activity diagram for which the access is to be controlled. The {role} specifies its value in form of the pair $(actor, role)$, whereby the *actor* is the actor in the activity diagram and the *role* likewise. The {right} also has its values in form of pair(s) $(role, right)$, where *role* is role, and *right* is for the right to access the protected resource. The *constraint* for the stereotype requires that the actors in the diagram only perform activities that they have appropriate rights for. The formalization of this can be explained as follows; for a given subsystem S and every actor A in S, and every activity a in the swimlane of A in the subsystem of S, there exist a role R making the value of {role} (A, R) and the value of {right} (R, a).

Tags	Type	Multipl.	Description
{protected}	state	*	protected resources
{role}	(actor, role)	*	assign role to actor
{right}	(role, right)	*	assign right to role

Table 4.3: Associated tags for «rbac»

«critical»

The stereotype of «critical» may be used on subsystems (i.e. models, packages) or objects (i.e. class(es)) to specify that the data in contained within the instances are critical in some manner. The corresponding tags are {secrecy}, {integrity}, {authenticity}, {fresh} and {high}. The tag {secrecy} is the name of the expression(s), attribute(s) or message argument variable(s) of the object to which the secrecy or confidentiality is to be protected. It is also possible to provide the name of the operation for which its arguments and return values are to be confidential or kept secret. The tag {integrity} consists of a pair of values (v, E), where v is the variable of the object to be protected, and E is a set of acceptable expressions that may be assigned to v. For {authenticity} you have the pair of values (a, o) where a stores the data whose authenticity should be provided and o stores the origin of that data. The {fresh} holds atomic data – data that is isolated and occurs once without interruption (thus preserving the *freshness* security policy) – that should be freshly generated.

These constraints are enforced by the constraints of «data security» stereotype, which by labels the subsystems that contain the «critical». The {high} tag holds the message(s) (i.e. the name(s)) that are supposed to be protected with respect to the security policy of *secure information flow*, which is furthermore enforced specifically by the «no down-flow» and «no up-flow».

Tags	Type	Multipl.	Description
{`secrecy`}	data	*	confidential data
{`integrity`}	(variable v, expression E)	*	integrity of data
{`authenticity`}	(data a, origin o)	*	authenticity of data
{`high`}	message	*	high-level messages to be protected
{`fresh`}	data	*	data to be freshly generated

Table 4.4: Associated tags for «`critical`»

«`Internet`», «`encrypted`», «`LAN`», «`wire`», «`smart card`», «`POS device`», «`issuer node`»

The stereotypes on links and nodes (figure 4.1) in deployment diagrams denote the various types of communication links and nodes. It is required that each link holds at most one of the listed stereotypes in UMLsec.

$$s \in \{\text{«wire»}, \text{«encrypted»}, \text{«LAN»}, \text{«smart card»}, \text{«POS device»}, \text{«issuer node»}, \text{«Internet»}\} \tag{4.1}$$

For every single adversary type A, UMLsec defines a function $Threat_A(s)$ from each stereotype in 4.1 to a set of strings $Threat_A(s) \subseteq \{delete, read, insert, access\}$ under the following conditions:

$$node \in s \implies threats_A(s) \supseteq \{access\} \tag{4.2}$$

$$link \in s \implies threats_A(s) \supseteq \{delete, read, insert\} \tag{4.3}$$

To further explain these conditions expressed as formulas (4.2 and 4.3), we can simply say that the $Threat_A(s)$ function specifies which kind of actions an adversary of type A ca apply to the nodes. To express it in a different manner, we can explain the first condition 4.2 as that if a given stereotype s (see 4.1) is for a node type, then the adversary type A can do harm by gaining *access*. The same applies for the second condition 4.3, except the adversary can do harm through the capability of *delete*, *read*, and *insert*.

`«secure links»`

Tags	Type	Multipl.	Description
{`adversary`}	adversary model	1	adversary type

Table 4.5: Associated tags for `«secure links»`

The `«secure links»` stereotype for labeling subsystems, has the purpose to ensure that the security requirements on the communication are met by the physical layer (e.g. deployment diagrams). Like many other stereotypes, `«secure links»` also specifies the adversary type using the {`adversary`}. To provide more detail on the constraints of the stereotype, we say that for a subsystem S and for each dependency d with the stereotype $s \in \{\texttt{«secrecy»}, \texttt{«integrity»}, \texttt{«high»}\}$ between subsystems/objects on different nodes n, m, we have a communication link l between n and m such that:

$$s(\texttt{«high»}) \implies threats_A^S(l) = \emptyset \tag{4.4}$$

$$s(\texttt{«secrecy»}) \implies read \notin threats_A^S(l) \tag{4.5}$$

$$s(\texttt{«integrity»}) \implies insert \notin threats_A^S(l) \tag{4.6}$$

To explain the three constraints enforced by the `«secure links»`, equation (4.4) states that for any dependency with the stereotype of `«high»`, the threat from the default adversary is supposed to be non-existent. Meanwhile the constraint equation (4.5), states that the atomic action/threat of read, should not exist as a threat for any dependency with the stereotype of `«secrecy.»` The last equation and condition (4.6) states that the atomic action of insert, should not exist as a threat for any dependency with the stereotype of `«integrity»`.

Example: Given the subsystem labeled with the stereotype of `«secure links»` in Fig. 4.3, and the default adversary on Fig. 3.3 on page 30. This model is in violation of the conditions for the `«secure links»` expressed in equation (4.5). Specifically the link between the `ClientNode` and `ServerNode` does not provide communication secrecy against the *default* adversary. According to the $Threats_{default}(Internet)$, the communication link between the client and server does not provide the required security level. The reason is simply that communication via Internet does not provide secrecy against adversaries (default). We therefore get the following:

$$s(\texttt{«secrecy»}) \implies read \in threats_{default}^{RemoteAccessDemo}(Internet)$$

To improve the model, so that it is not in violation to the condition expressed in equation

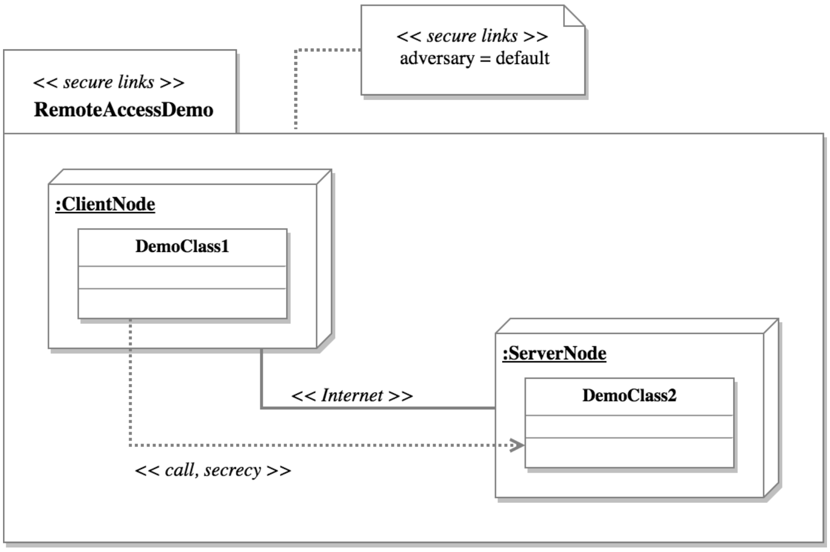


Figure 4.3: Example of «secure links» usage

(4.5), we can improve the RemoteAccessDemo subsystem with the following Fig. 4.4. The communication between client and server now provides secrecy against the default adversary, and therefore provides the necessary security level according to the $Threats_{default}(Internet)$. We therefore get the following expression:

$$s(\text{«secrecy»}) \implies read \notin threats_{default}^{RemoteAccessDemo}(encrypted)$$

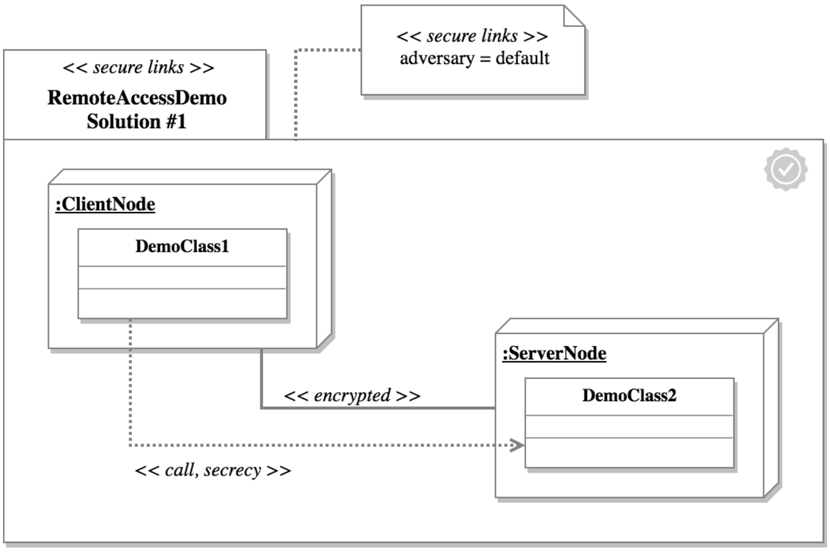


Figure 4.4: Example of valid «secure links» usage

«secure dependency»

This stereotype, meant for subsystems containing static structure (e.g. class diagrams) ensures that the dependencies of «call» and «send» across objects or subsystems respect the security requirements on the data communicated across them together with the {secrecy}, {integrity} and {high} tags associated with the «critical» stereotype. To elaborate further on the constraint for this stereotype; if there is a «call» or «send» dependency from an object or subsystem C to an interface I of an object or subsystem D, then we need to fulfill the following conditions:

$$msg(n) \in I \wedge msg(n)^C \in \{tags_{critical}\} \implies msg(n)^D \in \{tags_{critical}\} \tag{4.7}$$

where $tags_{critical}$ denotes the tags of either {secrecy}, {integrity} or {high}

$$msg(n) \in I \wedge msg(n)^C \in \{tags_{critical}\} \implies dep(stereotypes_{critical})^C \tag{4.8}$$

where $stereotypes_{critical}$ denotes the stereotypes of either «secrecy», «integrity», or «high», and *dep* for *dependency.*

To explain the two constraints enforced by the stereotype of «secure dependency» expressed in as conditions within the equations (4.7) and (4.8). Firstly the conditions states that for any message n contained in the interface I, if the message n appears in the tag of {secrecy}, {integrity} or {high} within the object (i.e. class) of C, then should also appear as a tag in within the object of D. The second condition (4.8) states that if the message n appears in the interface I also appears in the tags of {secrecy}, {integrity} or {high} within C, then the dependency should be stereotyped with the respective stereotype of either «secrecy», «integrity», or «high».

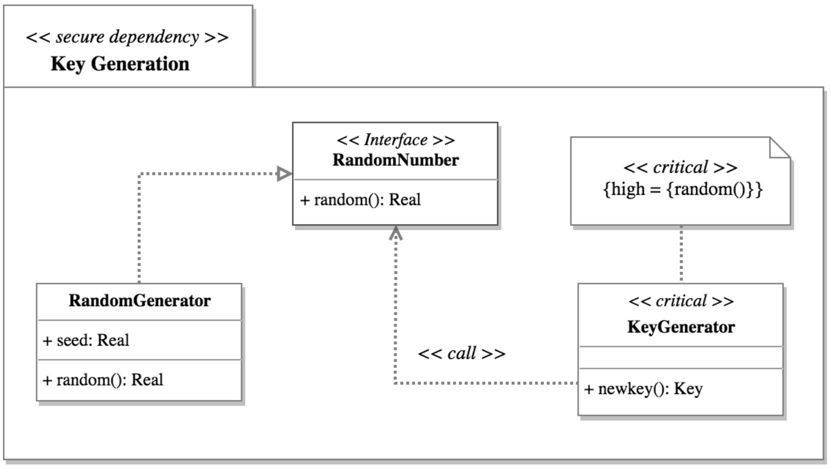


Figure 4.5: Example of «secure dependency» usage

Example: Given the subsystem labeled with the stereotype of «secure dependency» in Fig. 4.5. This subsystem provided by [Jürjens, 2005], show a key generation subsystem, labeled with the stereotype «secure dependency». This model is in violation of the conditions for the «secure dependency» expressed in equation both (4.7) and (4.8). This is because neither the «call» dependency nor the RandomGenerator provide the necessary security level required by the KeyGenerator. The *random()* message is required to be of high level by the KeyGenerator, but it is not required of equal level by the RandomGenerator. RandomGenerator is therefore required to have the {high} tag assoicated with the «critical» stereotype. Lastly the dependency of the «call» dependency is also required to be stereotyped «high» as expressed by (4.8). The proper use of the «secure dependency» stereotype is expressed by the Fig. 4.2.2.

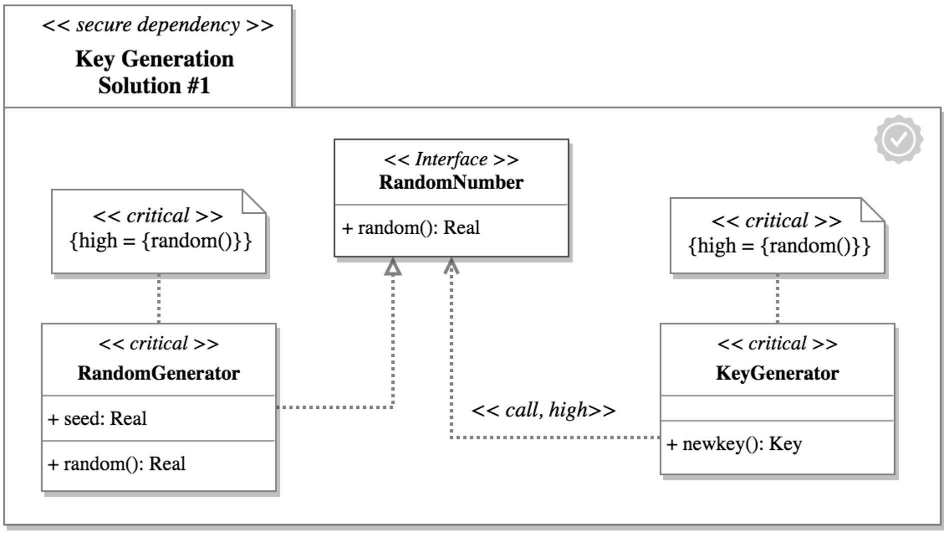


Figure 4.6: Example of valid «secure dependency» usage

«data security»

This particular stereotype ensures that security is enforced on the behavior level. Firstly, this stereotype for label subsystems has the following constraint: The behavior of the subsystem respects the security requirements given by the «critical» stereotype and associated tags, contained within a subsystem with respect to the threat scenario arising from the deployment diagram. This constraint is given by the following four conditions:

- ***Secrecy***: The subsystem preserves the secrecy of the data appointed by the {secrecy} tag against the adversary type A.
- ***Integrity***: The subsystem preserves the integrity of the data appointed by the {integrity} tag, against the adversary type A.

- ***Authenticity***: The subsystem preserves the authenticity of any value (data a, origin o). More specifically the authenticity of the attribute a, with respect to its origin against the adversary type A.

- ***Freshness***: Given a subsystem S stereotyped `«data security»`, the following conditions apply for any instance of a subsystem model or object model D within S, stereotyped `«critical»` for any value *data* associated with the {`fresh`}:
  data occurs within S *at most in:*

  i. *Static structure diagrams*: The instance of a subsystem or object model representing D contained in S.
  ii. *Activity diagrams*: The swimlanes belonging to D contained in S.
  iii. *Statechart diagrams*: The models parts of the behavior of D contained in S
  iv. *Sequence diagrams*: D's part of the connections in the digram contained in S.

In other words, it is sufficient for the data to be listed with a security requirement *one* of the object or subsystem instances contained within the `«data security»` subsystem to fulfill the aforementioned conditions.

The properties of the first three conditions (*secrecy, integrity and authenticity*) are then relative to the considered adversary type. Moreover in each of these cases, it is presumed that the initial knowledge of the default adversary (see Fig. 3.3 on page 30) not to contain the data values that according to the tags of the `«critical»` stereotype should be guaranteed *secrecy*, *integrity* or *authenticity.* Additional assumptions on the initial adversary knowledge can be specified, since the security requirements (*secrecy* etc.) cannot be achieved if the adversary already knows this data initially. So, in case the intended origin of the data in the {`integrity`}, {`secrecy`}, or {`authenticity`} refer to the expressions not locally known to the `«critical»` subsystem or object where these tags are applied, one can also associate these tags with the relevant `«data security»` stereotypes.

Another important note, associated with the conditions of the `«data security»` stereotype is that the considered system might be part of a larger system. The aforementioned conditions only assure that the component under consideration keeps the values received by the environment secret (or abide the other security requirements). Therefore we also need to make sure that the environment, specifically the rest of the system apart from the the component under consideration does not make these values available to the adversary (or violate the constraints of other applied security requirements). Note that the verification of the `«data security»` constraint is not so trivial, and is not completely covered within this thesis. Therefore for a more detailed explanation and examples, all the needed information can be found in the book "*Secure Systems Development*" by [Jürjens, 2002], which these security concepts where introduced.

Tags	**Type**	**Multipl.**	**Description**
{adversary}	adversary model	1	adversary type
{integrity}	(variable v, expression E)	*	integrity of data
{authenticity}	(data a, origin o)	*	authenticity of data

Table 4.6: Associated tags for «data security»

«guarded access»

This stereotype, intended for subsystems expresses that the object (i.e. class) that is stereotyped «guarded», should only be accessed through the objects specified by the tag {guard}, which in turn are attached to the «guarded» object.

«guard»

This particular stereotype is used to label objects (i.e. classes, interfaces) specifically in the scope of the «guarded access», that are supposed to be guarded. The stereotype has a tagged value {guard}, which defines the name of the corresponding object.

Tags	**Type**	**Multipl.**	**Description**
{guard}	object name	1	guard object (i.e. class or interface)

Table 4.7: Associated tags for «guarded»

4.3 Conclusion

This section will provide more details on how we can apply the well-formedness rules onto stereotypes, tags and constraints for constructing a profile to extend the metalanguage of UML. Hence, we now know which model elements we can apply the stereotypes, tags and constraints on, but we now proceed to looking specifically at diagrams.

4.3.1 Diagrams & Model Elements

The well-formedness rules of UMLsec help to showcases the model elements of UML (i.e. Class, Component, Package etc.) can be utilized for which exact stereotypes, and therefore which tags and constraints. The base classes, previewed by [Jürjens, 2005] and visualized on Fig. 4.1, essentially describe the existing UML meta-elements which the given stereotypes match. An example of this is the «Internet» corresponding to the base class of *Link*, which

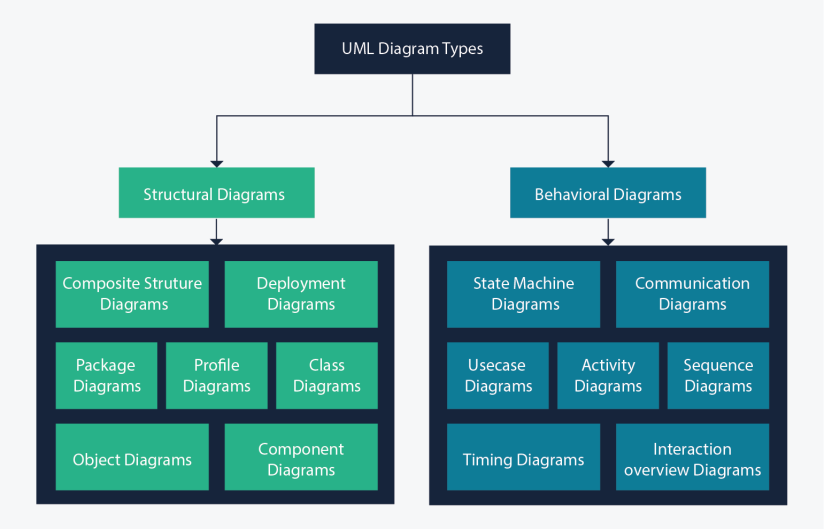


Figure 4.1: Categories of UML 2.0 Diagram Types. [Nishadha, 2012]

easily tells us which UML element the stereotype can be applied and labeled onto. This is the basis for how we construct the ThingMLsec profile by extending the UML-metalanguage using extension mechanisms based off the well-formedness rules of UMLsec.

UML modeling constructs can arguably be divided into two major areas and diagram types (i.e. Structural and Dynamic Behavioral), and so provide a structured view of the exact model elements or diagrams. We have also divided the diagrams into Structural and Behavioral diagrams types. It is notable to mention that there is no clear distinction between the constructs, but this is how they are often divided for convenience and guidance of use. The distinction of views that we choose to follow is the Fig. 4.1. Certain model elements like package, dependencies and communication paths are used for several types of diagrams, and therefore the various stereotypes can also be applied to several types of diagrams. Previously in this chapter we described the *base classes*. These are specified in the table overview of stereotypes, and we shall now explain the various stereotypes according to their diagram types below as summarized in Table 4.1.

Package

For the Package diagrams we can apply the stereotypes of `«fair exchange»`, `«rbac»`, `«secure links»`, `«secure dependency»`, `«guarded access»`, `«data security»`, `«no down-flow»` and `«no up-flow»`. Since both behavioral and structural diagrams can be contained within package diagrams, the aforementioned (at least some) can be applied either one of the diagram types or even both.

Table 4.1: ThingMLsec Stereotypes and UML 2.0 Diagrams

		«fair exchange»	«provable»	«rbac»	«Internet»	«encrypted»	«LAN»	«wire»	«5G»	«BLE»	«actuator»	«secrecy»	«integrity»	«high»	«critical»	«secure links»	«secure dependency»	«data security»	«no down-flow»	«no up-flow»	«guarded access»	«guarded»
Structural	Component	–	–	–	–	–	–	–	–	–	–	✓	✓	✓	✓	–	✓	–	–	–	–	–
	Deployment	–	–	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–
	Class	–	–	–	–	–	–	–	–	–	–	✓	✓	✓	✓	–	✓	–	–	–	–	✓
	Package	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	✓	✓	✓	✓	✓	✓	–	–
	Object	–	–	–	–	–	–	–	–	–	–	✓	✓	✓	✓	–	–	–	–	–	–	✓
Behavioral	Activity	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	Use Case	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	Sequence	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	✓	✓	–	–
	State Machine	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	✓	✓	–	–

✓= Applicable;

Deployment

For deployment diagrams, we use «secure links» to ensure that the security requirements (i.e. security assumptions) on the communication dependencies are supported by the physical description. Since deployment diagrams contain nodes, communication links and dependencies, we can use the UMLsec stereotypes specific for those model elements. For nodes we can use the «LAN», «smart card», «POS device» and «issuer node». However for this thesis and appliance to IoT some of the UMLsec stereotypes associated with nodes, were not aligned or relevant to this, so all stereotypes (i.e. for nodes) except for the «LAN» stereotype where excluded in the ThingMLsec profile.

For the communication links the UMLsec stereotypes of «Internet», «wire», «encrypted», «LAN» can be used for this diagram type. Furthermore to adopt the UMLsec approach for IoT and the use case we extended the stereotypes with «5G» and «BLE» in our profile. Since dependencies are used to model relationships between given nodes and eventually objects within the nodes, we can use the stereotypes related to dependencies which are «secrecy», «integrity» and «high». Lastly the «critical» can be applied to classes (i.e. objects), components or interfaces, and therefore can also be applied within deployment diagrams.

Component

For Component diagrams, we use the «secure dependency» to ensure that the security requirements in different part of a static structure diagram are consistent. To model the dependencies (i.e. relationships) between various components, we may use the given UMLsec stereotypes; «secrecy», «integrity» and «high». Like for deployment diagrams, we can also use the «critical» stereotype.

Class & Object

In terms of Class and Object diagrams, we also use the «secure dependency» for consistency within static structure diagrams, the same way as for Component diagrams. The Class and Object diagrams can for the most part use the same stereotypes, for as we previously described; an object is an instance of class, and they bot describe the static structure.

Activity & Use Case

As for the behavioral diagrams, there are very few steretypes that express the behavoioral constraints in UMLsec. At least compared to the structural diagrams. Firstly, for the Activity diagrams we can apply and enforce the stereotypes of «fair exchange», «provable» and «rbac». These security policies constraint the behavior of Activity diagrams.

Sequence & State Machine

The Sequence and State machine diagrams can be labeled with the same stereotypes. Both stereotypes express the secutiry policies of `«secure information flow»` and are `«no down-flow»` and `«no up-flow»`. Similar to the the other behavioral diagrams, these stereotypes essentially express the semantic behavioral constraints.

CHAPTER 5

UMLsec Profile Construction

We have now defined the rules with regards to the extension mechanisms and constructed our own profile shown in the figure 5.1. To summarize, the ThingMLsec is a profile diagram that operates at the metamodel level of UML by extending the base metaclasses of UML (e.g. Model, Node, Class etc.), with stereotypes as classes, marked with the `«stereotype»` label. The relationship between the metaclasses of UML and the stereotypes, are visualized with the help of the extension relation indicating which stereotype(s) the metamodel element/class is extending. The intention behind this is to use the ThingMLsec to label all standard UML 2.0 model elements, appearing as if it is a part of the UML metamodel.

5.1 Tool Support

The important part of constructing the UML profile is determining the tool to use for this purpose. In this section we look at the alternative tools and why they weren't used for constructing ThingMLsec.

The idea was always to use tools that allow us to complete our goal – i.e to define and apply a UML profile. And since the success of our goal determined so heavily on the capabilities of the tool, an important task was to find and compare the different tools available, and based on our needs, further select the most suitable tool. We chose to define and apply our ThingMLsec profile using Papyrus UML, but before doing so we weighed the following requirements[1] that the tool needed to meet.

[1] *The order of the requirements are arranged by importance. (RQ1 = most important etc.)*

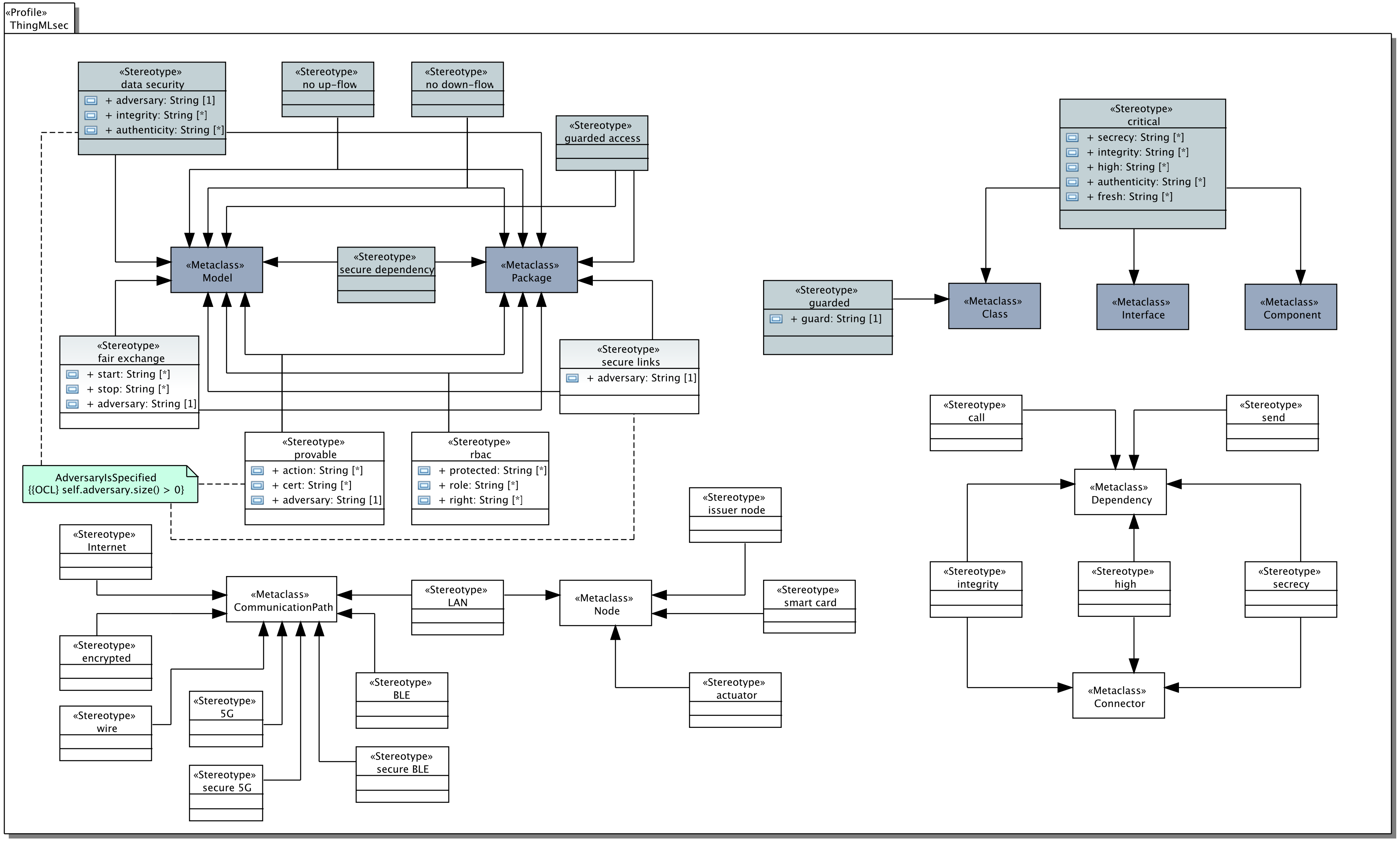


Figure 5.1: The Complete ThingMLsec Profile

Req.	Requirement Description
RQ1	Requires UML2 support.
RQ2	Requires UML profile support.
RQ3	Requires OCL support.
RQ4	Requires tool to be open source and free.
RQ5	Requires possibility for Eclipse IDE integration.

Table 5.1: Requirements for the selection of UML tool

5.1.1 Papyrus

Papyrus[2] is an Eclipse graphical editing tool for UML2. It is free and open source, and also available as an Eclipse-plugin as part of the Model Development Tool (MDT) project. One of it's greatest feature is that it provides extensive support for UML profiles. The tool includes all the facilities for defining and applying UML profiles in a rich and efficient manner, and it succeeds in implementing for UML as complete as possible (e.g. supporting latest diagram types). The tools also provides powerful tool customization capabilities similar to DMSL-like metatools. Papyrus enables us to use a general purpose language (i.e. UML2), but also DMSL approaches through profile mechanism (i.e. ThingMLsec).

5.1.2 Alternatives

There are a lot of commercial and open-source tools that are available for UML modeling, and amongst these we have ArgoUML, MagicDraw, Papyrus, Modelio etc. During this subsection, we look at some of the most popular modeling tools today, and assess how they fulfill the given requirements.

MagicDraw

MagicDraw[3] is a commercial, award-winning business process, architecture, software and system modeling tool. It is designed for many different software users (e.g. designers, architects, developers etc.), for many different purposes. MagicDraw is a dynamic and versatile development tool with extensive support for many programming languages. It comes in different editions for specific users, enriched with many features, and through it is commercial, like most commercial software it can also be downloaded as a trial version for a limited amount time. MagicDraw fulfills all the requirements we have, like support for UML2 modeling, OCL and the possibility to be integrated with Eclipse.

[2] https://www.eclipse.org/papyrus/

[3] https://www.nomagic.com/products/magicdraw{#}overview

Modelio

Modelio[4] is an open source modeling environment which has the possibility to be extended through models in order to add functionalities and services. As many modeling tools today, it supports UML2, including other standards like BPMN2, SysML etc. However the tool does not support OCL, but it does allow to add constraints as part of UML to achieve a similar outcome. Lastly the tool has the possibility to be integrated with Eclipse.

ArgoUML

ArgoUML[5] is an open source modeling tool, that supports all standard UML 1.4 diagrams. The tool runs on any Java platform and is available in 10 languages. The tool allows for profile support with even some profiles already provided by the tool. ArgoUML also supports constraints using OCL. The integration support (Eclipse) is not so clear to see, from the website, or the tool itself.

During the use and testing of the tool, we noticed that the UI seemed very outdated, indicating that the tool had not been maintained for a while. This would also explain the lack of UML 2.0 diagrams. Although the tool was stable under use, the fact that it had not been maintained and the last stable version was last updated in 2011, is quite concerning. This implies that the tool will probably no longer be safeguarded to new threats, or provide new features to improve the usability and such. And even if we could integrate the tool with Eclipse IDE, this remains a potential problem – i.e. to integrate an outdated software with a well maintained one.

5.1.3 Conclusion

We chose to use Papyrus for defining and applying UMLsec, because of the requirements we had on beforehand, and that we needed the tool to satisfy for us, to be able to complete the task. During the assessment process we found that some of the other tools had many useful features beyond our requirements, but some sadly did not meet the requirements needed to be able to implement ThingMLsec.

The summary of the results are depicted in the table shown in Fig. 5.2. The table shows not only the requirements we initially set prior to the selection process (i.e. RQ1-RQ5), but also other requirements that where added to further determinate which tool would ultimately suit the purpose. RQ6 and RQ7 are based on an extensive comparison case study by [Safdar et al., 2015], and finally RQ8 shows how well the tools have been maintained and improved lately.

[4] `https://www.modelio.org/about-modelio/features.html`

[5] `http://argouml.tigris.org/features.html`

The first requirement RQ1 is not met but ArgoUML, possibly due to lack of new updates of the tool. And since this was the most important requirement, we had to eliminate the tool from the selection process. The second requirement, RQ2 was met by all tools, but Modelio provides UML profile support in form of modules. While this is not necessarily a drawback it makes the method less intuitive for users already acquainted with most UML modeling tools. The third requirement RQ3 was not met by the Modelio tool at all, which was the main reason for elimination from selection. The fourth requirement RQ4 was not a crucial requirement, but an open source and free tool which would provide a lower threshold for most (i.e. organizations and other users) to try out the tool and approach. The only tool that did not fulfill this requirement was MagicDraw, but this was not major delimiting factor or main reason for not selecting the tool, since the tool has a 30-day trial version and provides many great features.

Table 5.2: Comparison of UML tools

Requirement	**Papyrus**	**MagicDraw**	**Modelio**	**ArgoUML**
RQ1: Requires UML2 support.	●	●	●	○
RQ2: Requires UML profile support.	●	●	◐	●
RQ3: Requires OCL support.	●	●	○	●
RQ4: Requires tool to be open source and free.	●	○	●	●
RQ5: Requires possibility for Eclipse IDE integration.	●	●	●	-
RQ6: Effort required in use.	★★☆	★★☆	-	-
RQ7: Learnability.	★☆☆	★★★	-	-
RQ8: Last stable version	2017	2017	2018	2011

● = Provided; ◐ = Partially Provided; ○ = Not Provided;

5.2 ThingMLsec Profile

ThingMLsec is the a UML profile diagram, which defines extensions to the UML metamodel, derived from the well-formedness rules of UMLsec together with added concepts for IoT systems development in order to adopt the metamodel to a specific domain. This profile diagram is visualized in Fig. 5.1, where the UML metamodel is extended via stereotypes, which further can have properties, referred to as tags or tag definitions. The only thing that is lacking is the further application of formal constraints in OCL, which would have allowed to check the validity and the consistency of any given model created using the profile.

The benefits of using the ThingMLsec Profile:

B1: ***Expressiveness***; we can enforce security requirements to any UML2 model available, using a well-known and commonly used approach (ie. extension mechanisms).

B2: ***Selection freedom***; we have the ability to model security using any UML modeling tool since most tools support UML profiles.

B3: ***Integration of modeling and analysis***; by adding OCL constraints in the ThingMLsec profile, we can not only model, but also verify our models to eventually improve the security of our models.

B4: ***Improved maintainability***; the ThingMLsec as profile is defined as a package diagram that can be constantly edited, improved and extended with new stereotypes and OCL constraints.

B5: ***Tool support***; we can use, develop and or/integrate the ThingMLsec profile with other tools, (i.e. by the help of Eclipse-plugins).

5.2.1 OCL

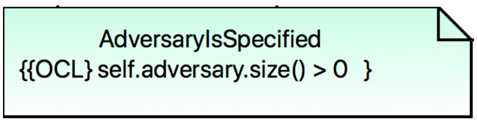
We can use the a profile to define constraints. Our ThingMLsec profile contains a condition that is denoted as part of UML called Object Constraint Language (OCL) depicted in Fig. 5.1. OCL is essentially a textual general-purpose formal language adopted as a standard by the Object Management Group (OMG) [Cabot and Gogolla, 2012]. OCL is a key element of Model-Driven Software Engineering (MDSE). With OCL, we have the ability to define new domain-specific languages using well-formedness rules, or express model transformations or code-generation templates. It is used to define several kinds of expressions to complement the information of UML models, and specifically complementing the models with model queries, manipulation and specification requirements. Furthermore we can say that OCL is a *typed*, *declarative*, and *side-effect* free *specification* language.

To further clarify, OCL is *typed* in the sense that each expression evaluates to a type (either pre-defined or contained type in the model that OCL is being used) and must conform to the rules and operations of that type. With *declarative*, in the sense that OCL does not include imperative constraints like assignments, which is essentially describing what you want to do and not how you want to do it. *Side-effect free*, in the sense that the OCL expressions constrain the state of the system, but not alter it. Lastly, it is a *specification* language in the sense that OCL does not include any implementation details or guidelines.

An example of how we can use OCL is by constraining elements in the UML metamodel. This is done in the sense that we can constrain all Class instances to be active, or that all instances of Class are allowed at most one operation. The drawback is that for the constraints to be automatically validated, we need to first rewrite them into a language that the Papyrus tool is able to understand. For the specific tool of Papyrus this language is usually Java [Gérard, 2011]. Papyrus supports two ways of validating constraints that are applied to models., and they are as following:

Profile Embedded OCL Constraints

This first method is a simple and straightforward way of defining constraints related to the Papyrus tool, where we can define constraints directly into the profile, as expressed in Fig. 5.1. This method of defining a constraint allows us to define the constraint within the profile itself, in contrast to defining and embedding constraints into a plugin. In the aforementioned figure, we can see a snippet of a constraint that is taken from the complete ThingMLsec profile in Fig. 5.1. In this constraint, we have created a Constraint Node, which is available as part of the Palette (i.e. quickly accessible model elements) for the package diagrams. The constraint, named *"AdversaryIsSpecified"* simply states that the adversary attribute, to which it is connected to, must be not be empty. In other words that the adversary , should not be of empty type *string*. More specifically, the constraint expresses the condition stating that the stereotypes capable or allowed to specify the adversary type, *must* specify adversary type. In Fig. 5.1, we can see that this constraint is referenced to the elements (i.e. stereotypes) of `«provable»`, `«data security»`, `«secure links»` and `«fair exchange»`.



AdversaryIsSpecified
{{OCL} self.adversary.size() > 0 }

Figure 5.1: ThingMLsec OCL Profile Constraint

Plugin Embedded OCL Constraints

This method on the other hand, is more complicated but yet more powerful than the latter. Firstly it requires an additional extension for the Papyrus Modeling tool; *Papyrus DMSL Validation* [Ericsson and Letavernier, 2018]. This method in particular allows us to generate a plugin from a profile that embeds the constraints, expressed in either OCL or Java. Firstly, OCL constraints are embedded into the plugin.xml file, while the Java constraints can be directly complied into code. The constraints are grouped into a category that can be included into the validation or not. Lastly, it is worth noting that one should not embed the constraints both in the profile and the plugin. In other words, it is not recommended to select both options.

Previously in Chapter 3, we distinguished between three different ways of adding security relevant information to diagrams. These specific groups, also defined as three types of stereotypes; (i.) *security* assumptions that describe the physical level of the system, (ii.) *security requirements* that describe the logical structure of the system or specific data values, and lastly (iii.) *security policies* that specify what parts of the system should obey. While security assumptions and requirements simply add the information to the model and are applicable to various model types, the security policies are have constraints that have to be fulfilled by the diagram to carry the stereotype (e.g. `«secure links»`, `«secure dependency»` etc.).

The idea specifically with embedding the OCL constraints into a plugin, we could automatically verify the models with respect to the applied stereotypes and constraints. However, we did not have sufficient time for this task, since understanding and using the tool had priority.

Part IV

Application

CHAPTER 6

The Application of ThingMLsec to ALCCS

The main idea is that once we demonstrate the use of ThingMLsec profile onto both a few structural and behavioral diagrams, in order to express the capabilities of UMLsec and security specification on models. Notice that the quality of the models is not the focus of the thesis. They are from the project of SCOTT, and the main focus for the thesis and this particular chapter is foremost the applicability of ThingMLsec onto the provided models and views.

The application of ThingMLsec will be demonstrated in the following way; we shall first *apply* the stereotypes and then *verify* the constraints for the structural view and models, and after that onto the behavioral view and models.

6.1 Proof of Concept

Before we start applying and verifying with ThingML, we shall first describe the provided use case and scenario that will be used. This is specifically the behavioral diagrams (i.e. use case and activity).

The "*Use Case specification and System Architecture for Assisted Living*" by [Brandsma, 2017], provides two distinct use cases, where one of them is "Patient Trend Monitoring" and the other "Emergency Handling". The overall design and architecture will be for the entire system, but the modeling of specific scenarios for behavior diagrams, will be for this use case.

6.2 Use Case Description

We wish to demonstrate the use and appliance of UMLsec using the use case of the *Assisted Living and Community Care* (ALCCS) and specifically the use case of *Emergency Handling.*

6.2.1 Emergency Handling

The main idea with *trust delegation* is that we can locate the most appropriate person nearby to give immediate care and providing that exact person access to enter the residents' (e.g. elderly users) home. In this case is by accessing a wirelessly controlled lock on the front door. In addition to this, to also provide relevant data about the individual. [Brandsma, 2017] depicts this in the form of Fig. 6.1.

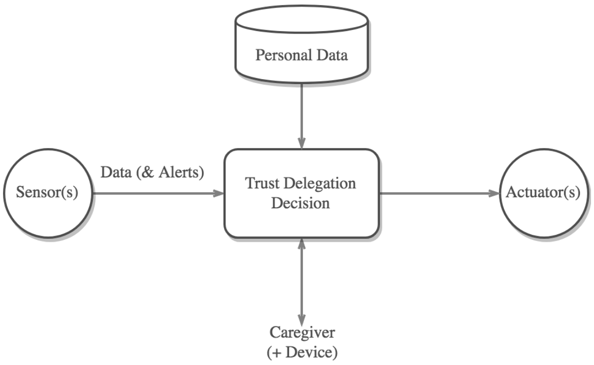


Figure 6.1: Emergency Handling Use Case [Harrand et al., 2016]

Emergency handling is therefore about arranging the quickest and most appropriate care when a patient experiences an emergency. To further clarify this we can envision the following situation: An elderly person living alone, possibly suffering from dementia (e.g. mild), may fall and experience trouble getting back up again, or eventually incapacitated in some way. This situation could therefore be detected by (e.g.) using a panic button, a fall detector, or by tracking the persons location and movement.

We will look further into a specific use case scenario for emergency handling, which is the "Call for Help" scenario. This use case is described in full in table 6.1.

Use Case	WP21 Assisted Living and Community Care
Scenario:	Call For Help
Primary Actor:	Resident

Actors and Interests:

A1 Resident: Wants to get help from the most appropriate trusted person in case of emergency.

A2 Caregiver: Wants to get information about the emergency and access to the facilities to be able to resolve it.

Description: This scenario describes a procedure of calling and dispatching the most appropriate Caregiver for help in case of an emergency situation.

Trigger: The Context Reasoning System triggers the Call for Help (i.e. a panic button push by the Resident is handled through the Context Reasoning System as well.) The Context Reasoning System has a prepared list of Caregivers based on the contexts of Resident and potential Caregivers.

Main Success Scenario:

1. The Edge System receives a Call For Help from the Context Reasoning System, retrieves the prioritized list of Caregivers and sends an emergency signal to the most appropriate Caregiver (i.e. top of the list)
2. The selected Caregiver receives the emergency signal, decides to accept the call and to initiate a voice connection with the resident.
3. The Resident does not respond or the Resident responds and indicates that he/she is an emergency situation (or setting up a voice call fails.)
4. The Caregiver indicates whether he/she can handle the emergency himself/herself.
   a. If the Caregiver can handle the emergency: *Execute Sub-scenario S1.*
   b. Else: *Execute Sub-scenario S2.*

Sub flows:

S1. The selected Caregiver can handle the emergency:

   a. The Edge System sends credentials of the responding Caregiver to the SmartLock of the frontdoor of the Resident's house.

   b. The Smart Lock updates the access credentials with the ones from the responding Caregiver.

S2. The selected Caregiver can't handle the emergency:

   a. The Caregiver provides additional context information based on his/her voice call conducted with the Resident.

   b. The Analyze Behavior scenario is invoked to recompute a prioritized of Caregivers and subsequently re-invoke this "Call For Help" scenario updated list.

ALTERNATE/EXCEPTION FLOWS:

A1. (*step 2*): The selected Caregiver decides to decline the call:

   a. The Edge System will select the next Caregiver from the prioritized list and send an emergency signal to him/her.

   b. The flow continues at *step 2 for new Caregiver*.

A2. (*step 2*) The selected Caregiver accepts the call, but a voice connection with the Resident is not needed before heading over there:

   a. The flow continues at *Sub-flow S1.*

E2. (*step 3*) The Caregiver and the Resident agree this is a false alert and no further action follows.

INPUT DATA:	List of credentials for Smart Lock Prioritized additional list of Caregivers and potential Caregivers' contexts.
OUTPUT DATA:	Updated list of credentials for the Smart Lock system. Potentially additional context information about the Resident.
INFRASTRUCTURE:	Smart Lock Connected to Edge System with Internet access, Caregivers Wearable with internet access.

Table 6.1: Use Case Scenario 1 Description, WP21 ALCCS

6.3 Structural View

In this section we will apply and verify the stereotypes of our ThingMLsec onto two types of structural models (Deployment and Class diagrams) provided by [Brandsma, 2017] (remodeled using Papyrus). Furthermore we shall go through the verification of the various constraints that are applied together with different stereotypes.

6.3.1 Deployment Diagram

The improved deployment diagram and proposed solution for ALCCS depicted in Fig. 6.2 from [Brandsma, 2017, p. 16], shows the model contained within a package diagram stereotyped `«secure links»`. To formulate and enforce the various security requirements, assumptions and policies, the communication links of Internet, encrypted, LAN, secure BLE and secure 5G, along with the dependencies of secrecy, integrity, call and send are used to connect the various nodes and artifacts contained within the nodes. We previously mentioned the stereotypes that are used especially for deployment diagrams, and can be summarized as shown in Fig. 6.1. In detail, the diagram in Fig. 6.2 consists of nine nodes; Elderly Wearable, Web Server, Cloud Server, Smart Phone, Access Point/Router, Sensor Gateway, Sensors, Wireless Data Aggregator (WDA) and Smart Lock. These nodes are provided by the high level architectural description and view by [Brandsma, 2017] for ALCCS. The various nodes communicate via the five assorted connections; Internet, LAN, secure 5G, secure BLE and encrypted.

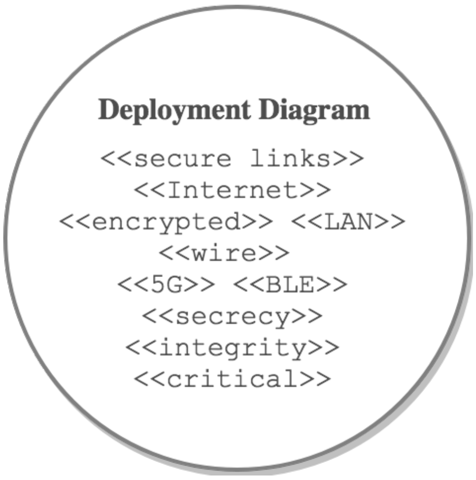


Figure 6.1: Stereotypes for Deployment Diagrams

Application

The idea is that the Assisted Living and Community Care System (ALCCS) should preserve the confidentiality and integrity of the data (i.e. the lists of authorized users, caregivers and elderly users, the location(s), sensor data etc.), that is being transmitted between the differing nodes and components/artifacts contained within the nodes. Therefore the corresponding dependencies stereotypes «secrecy», and «integrity» are included, and for usage, both the «call» and «send» stereotypes are also included.

The «call» dependency, utilized twice in the entire diagram, is first included between the communication between the Elderly Wearable and the Web Server nodes, where we assume that the *call* from the wearable invokes the operation of "Call for Help" depicted in the use case scenario on Fig. 6.1 on page 65. The other «call» dependency included in the deployment diagram is between the Smart Phone (Caregiver) and the Elderly Wearable (specifically the Audio Module artifact) over 5G cellular network in order to invoke the operation of communicating with the elderly. The other dependency type that is used – the «send» dependency is most used in the diagram, for example between the communication of Smart Lock and WDA, Sensors and Sensor Gateway etc., in order to specify the nature of the communication between the various nodes. This stereotype and specific dependency type is used to specify that the communication, where one node sends messages containing signals/data that the specific target receives. In other words, the «send» dependency is used in the diagram to express and depict the bidirectional nature of the communication.

Verification

At this point, we have labeled and applied the various stereotypes onto the specific model elements contained within the diagram to enforce the security requirements, and the diagram itself stereotyped «secure links». The next step is to verify the stereotypes that constrain the structural design in our diagram against the *defaultIoT* adversary that we modified and defined in Fig. 3.5 on page 33. Furthermore, given the constraints defined by [Jürjens, 2005] and listed as conditions (4.4), (4.5) and (4.6) on page 43, we can conclude to the following:

$$s_0(\texttt{«high»}) \implies threats^{ALCCS}_{defaultIoT}(links) = \emptyset$$

$$s_1(\texttt{«secrecy»}) \implies read \notin threats^{ALCCS}_{defaultIoT}(links)$$

$$s_2(\texttt{«integrity»}) \implies insert \notin threats^{ALCCS}_{defaultIoT}(links)$$

where:

$$links \in \{Internet, encrypted, LAN, secureBLE, secure5G\}$$

The *links* essentially denote the various stereotypes on the communication links of contained in our development diagram. The constraints are not violated because they fulfill the aforementioned constrains for the «secure links» stereotype. More specifically, because all the communication links (i.e. Internet, LAN, encrypted etc.) between the various nodes provide the needed security levels according to the $Threats_{defaultIoT}(links)$ scenarios, thus ensuring the security requirements on communication dependencies are supported by the physical situation.

6.3.2 Class Diagram

The improved class diagram and proposed solution for ALCCS depicted in Fig. 6.3 by [Brandsma, 2017, p. 17] shows the model contained within a package diagram stereotyped «secure dependency». The stereotypes connected to class diagrams for enforcing security are depicted in Fig. 6.4. The diagram consisting of a total of 16 objects (classes, subclasses, interface etc.), some stereotyped with «critical», and the relationships between the classes stereotyped with «call», «high», «secrecy» and «integrity». Some of the notable classes and objects include the Wireless Data Aggregator (WDA), WSAN Gateway, Decision Engine, Interface, SAC Objects, Actuator, Sensor etc.

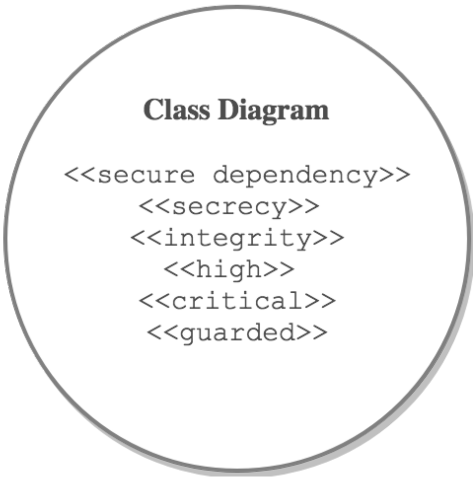


Figure 6.4: Stereotypes for Class Diagrams

Application

For the dependencies between classes and objects, we have chosen to label the relationships using the dependency type of *call*. This is not necessarily the dependency type to be strictly used for class diagrams, since we could equally have achieved the same effect with the *send* or the more general *use* dependency. The incentive behind the use of the «call» dependency is that the classes/objects making use of the connected classes/objects are making "calls" to make use of their services (i.e. methods), and therefore to invoke a specific operation or behavior, and hence the *call* stereotype.

In regards to the class diagram of the ALCCS, we also decide that the system should preserve the confidentiality, integrity and protection (to ensure secure information flow) to the data (i.e. resources, protocols, roles etc.) on the messages (i.e. the methods of *manageDevices*, *notfiyCaregivers*, *commision* etc.) Consequently, in order to label objects or classes containing data that is critical, we use the `«critical»` stereotype. Thereafter we use the corresponding tags to further specify in detail the security properties using {`secrecy`}, {`integrity`} and {`high`}.

Verification

Given the class diagram for ALCCS, we can state that the diagram has no interfaces connected with the stereotyped objects or classes (i.e. `«critical»`), thus invoking the exception rule for the `«secure dependency»` constraints explained in Eq. 4.7 on page 45. Since we have at this point labeled and applied the various stereotypes onto the model elements constrained within the ALCCS class digram, we now aim to verify the validity and integrity of the model with regards to the `«secure dependency»` constraints.

The provided specification and model in Fig. 6.3, does not violate the `«secure dependency»` because any two connected objects/subsystems (e.g. classes of SAC Objects and WDA, WSAN Gateway and SAC Objects, WDA and Interface etc.) within the model altogether respect the security requirements on the data that is communicated across them.

More precisely, we can say that the model fulfills the requirements specified by the constraints. Primarily for any two given classes/objects; e.g. SAC Objects and WDA, the security level for the *resources* data expressed specifically by the {`integrity`} and the {`secrecy`} tags is met by both objects. To express this in a different way, we can say that the *resources* in SAC Objects whose confidentiality and integrity is supposed to be protected, by the tags {`integrity`} and {`secrecy`}, are also guaranteed to be of the same security level and equally protected by the WDA object. We also see that for the SAC Objects class, in addition we wish to ensure the confidentially of the *location* data, hence the {`secrecy`} tag, and protect the message of *communicate* with respect to secure information flow. For the second constraint, which focuses on the dependencies between objects and subsystems, we can see how both dependencies are stereotyped with the respective tags `«secrecy»`, `«high»`, `«integrity»` and the usage dependency of `«call»`. therefore we can ensure that for any message that is exists in the tags of {`secrecy`}, {`integrity`} etc. in the SAC Objects, the dependencies are stereotyped respectively with the `«secrecy»`, `«integrity»` etc. Finally, by fulfilling these two constraints, the validity and integrity of the model is ensured with respect to the `«secure dependency»` constraints for the given stereotype.

We can observe this same pattern applied for any other connected objects within our model. For instance, the same applies for the WDA and Interface, where within the Interface, we wish to ensure the confidentiality and integrity of the data that is *location* and *resources*, and therefore the {`secrecy`} and {`integrity`}. Thereby we ensure that the security level is maintained for all connected classes and objects. The same applies for the dependencies on the two classes.

6.4 Behavioral View

In this subsection we further apply and verify the stereotypes of our ThingMLsec onto two other types of behavior models (Use Case and Activity diagrams) also provided by [Brandsma, 2017] (remodeled using Papyrus).

6.4.1 Use Case Diagram

The use case specification and system architecture of ALCCS by [Brandsma, 2017, p. 22] specifies four distinct use case scenarios for the system. The specific use cases depicted in Fig. 6.2 represent the various functionalities of the described environment [Brandsma, 2017, p. 22]. The four use cases are *call for help*, *analyze resident behavior*, *enter area* and *update resident location*. The specific use case and use case scenario that we will further look into is *call for help*. This means that we shall explore this specific description of the functionality from a different type of view – the activity view.

Use Case Diagram
<<fair exchange>>

Figure 6.1: Stereotypes for Use Case Diagrams

Application

For this use case diagrams, there is a very limited number of stereotypes that can be applied to enforce security, as depicted by the Fig. 6.1. This is mostly to the uncomplicated, that comes with expressing logical behavior without revealing the internal structure of the subject [Rumbaugh et al., 2010]. The slightly modified use case diagram of ALCCS therefore shows the model contained within a package diagram stereotyped «`fair exchange`». This specific stereotype used in use case diagrams represents the security requirement that any transaction should be performed in a way that prevents both parties from cheating. It is worth noting that [Jürjens, 2005] states that the stereotype applied to this diagram type hold an informal meaning. He further explains how the stereotype mostly serves as an example of how security requirements – as stereotypes – can be used in other diagrams types.

Verification

However with regards to the applied stereotype of «fair exchange», we wish to express that "cheating" in any form – mostly related to the service provider or context Reasoning System – can have fatal consequences. For instance if the system or service provider in any way neglects to respond to the *"call for help"* from the elderly user, the consequences can be fatal. Finally there is little to be verified in terms of this stereotype due to the informality of the security requirement, and the lack of specific constraints related to its use.

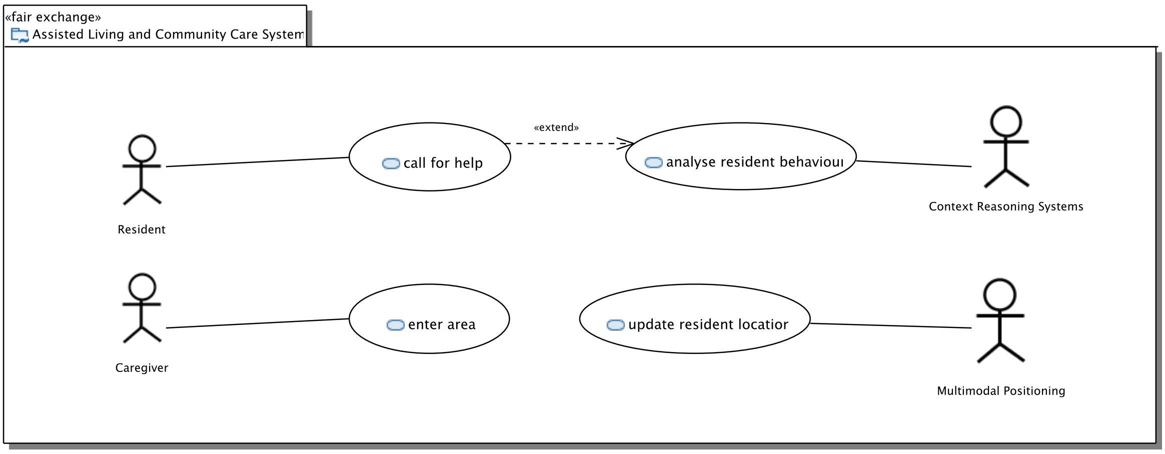


Figure 6.2: ThingMLsec Use Case Diagram of ALCCS

6.4.2 Activity Diagram

The use case specification and system architecture of ALCCS includes an Activity diagram of the "Call for Help" scenario depicted in Fig.6.2 and descriptively in Table 6.1 on page 65, from [Brandsma, 2017, p. 25]. The activity diagram in 6.5, in contrast to all the previous remodeled diagrams of ALCCS contains no ThingMLsec stereotypes. Due to limitations of the Papyrus tool and specifically lack of the Package or Model elements that are required to label security policies onto activity diagrams defined by our ThingMLsec. This is addressed and discussed in more detail in Chapter 7. Therefore to be able to apply the stereotypes that we needed to specify the security requirements for the activity diagram, we included a Model diagram that separately represents the unlabeled Activity diagram in Fig. 6.5 and applied to our the stereotype and associated tags there instead. This is the model depicted in Fig. 6.4.

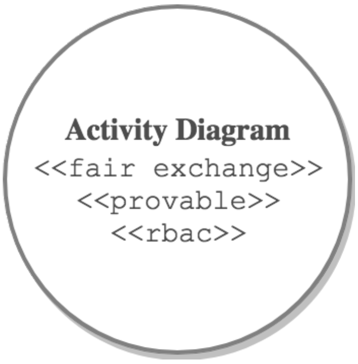


Figure 6.3: Stereotypes for Activity Diagrams

The improved and proposed solution for therefore for the activity diagram of ALCCS shows the model stereotyped «rbac». It is worth noting that the we could have used other stereotypes or a combination of several stereotypes shown in Fig. 6.3. The «rbac» expresses the security policy of access control and specifically Role-Based Access Control (RBAC). Firstly, from the activity diagram we can see the activities for the "Call for Help" scenario. The diagram is partitioned into five partitions; Resident, Residents Wearable, Edge System and Caregiver.

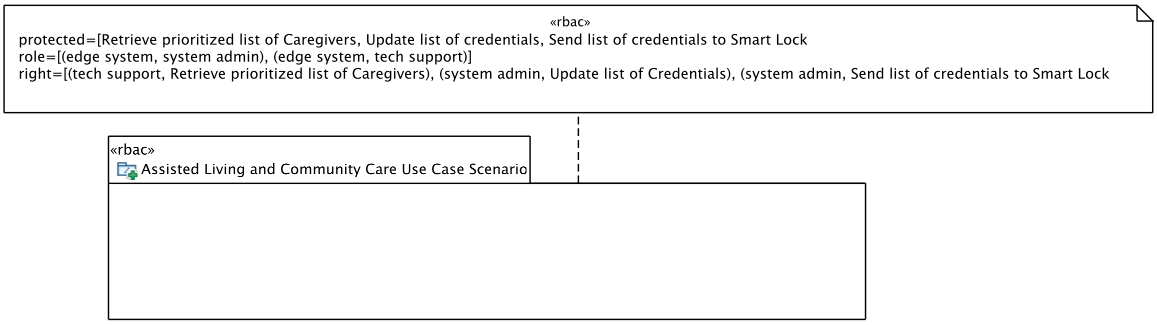

«rbac»
protected=[Retrieve prioritized list of Caregivers, Update list of credentials, Send list of credentials to Smart Lock
role=[(edge system, system admin), (edge system, tech support)]
right=[(tech support, Retrieve prioritized list of Caregivers), (system admin, Update list of Credentials), (system admin, Send list of credentials to Smart Lock

Figure 6.4: ThingMLsec stereotyped Activity Diagram for ALCCS

Application

For the Activity of "Call for Help", we wish to ensure and enforce role-base access control in order to express that only legitimate parties have access to the security-relevant or protected resources of the system. With RBAC we want to specifically make the permissions to the Edge System manageable, since we assume that this is a large system with many different users (tech. support, system administrators, supervisors etc.). So instead of users, we have *roles* that relate to their function within, and we assign the permissions to those roles.

With regards to the diagram, and specifically the partition of the Edge System, we assume there are several actors that are responsible for and perform the various activities within that particular swimlane. The corresponding tags for the «rbac» are {protected}, {role} and {right}.

Firstly we assume that the protected resources within the diagram are: (1) "Retrieve prioritized list of Caregivers", (2) "Update list of credentials" and (3) "Send list of credentials to Smart Lock". All these activities are contained within the swimlane of Edge System. Hence the {protected} expressed as following:

```
{protected = "Retrieving prioritized list of Caregivers", "Update list of
credentials", "Send list of credentials to Smart Lock" }
```

Furthermore, we assume that the Edge System has particularly a Technical Support responding to the call for help from the Elderly. He/she essentially deals with the lower-level and less important day-to-day activities. This is provided that this particular activity is not performed automatically. The other user of the Edge System is assumed to be the System Administrator with a more high-level responsibility and essentially deals with the more complex activities of the system. In other words, central to the "Call for Help" scenario and the Edge System, we have at assumed least two roles (1) Technical Support and (2) System Administrator. Note that the naming and accuracy of these two roles is insignificant, and merely provide details for the demonstration of applying the «rbac». Therefore we can assume that we have the following {role} tag:

```
{role = (Edge System, Technical Support), (Edge System, System Administrator)}
```

Lastly, we assume that the roles have specific rights to the protected resources. For the role of Technical Support, we assume that he/she should have the right to "Retrieve prioritized list of Caregivers" in this specific scenario, and the System Administrator should be able to "Update list of credentials" and "Send list of credentials to Smart Lock" within the same scenario. The corresponding {right} tag expresses this in the following manner:

```
{right = (Technical Support, Retrieve prioritized list of Caregivers), (System
Administrator, Update list of credentials), (System Administrator, Send list
of credentials to Smart Lock)}
```

These assumptions therefore describe the tagged values depicted in Fig. 6.4 with the «rbac» stereotype. Depending on the intention of the user, one can freely specify the protected resources and roles, but they should respect the associated constraint that we shall explain below.

Verification

Given the «rbac» stereotyped activity digram, we established the constraint for the stereotype during Chapter 4 under the well-formedness rules. The constraint essentially requires that the actors in the activity diagram only perform activities which they have appropriate rights.

In our model, we can observe the protected resources of "Retrieving prioritized list of Caregivers", "Update list of credentials" and "Send list of credentials to Smart Lock". The Edge System, using the roles of Technical Support and System Administrator respectively have the appropriate permissions to. Therefore we can conclude that the diagram is correctly labeled, and that the constraint for «rbac» is not violated.

Lastly, it is important to mention that we could visualize the role-based access control within the activity diagram much better by separating the Edge Systems partition into two (at least) swimlanes for the aforementioned actors, in order to clearly visualize the separated activities and protected resources based on the actors that actually are responsible for performing those specific activities. We mention this, because currently the protected resources are all contained in one partition of Edge System, while their permissions are most likely performed by different actors within the Edge System and that are not visualized within the diagram. This is merely a suggestion, and that has not taken into account the reasons that the project and system architects of SCOTT had in mind while specifying these models.

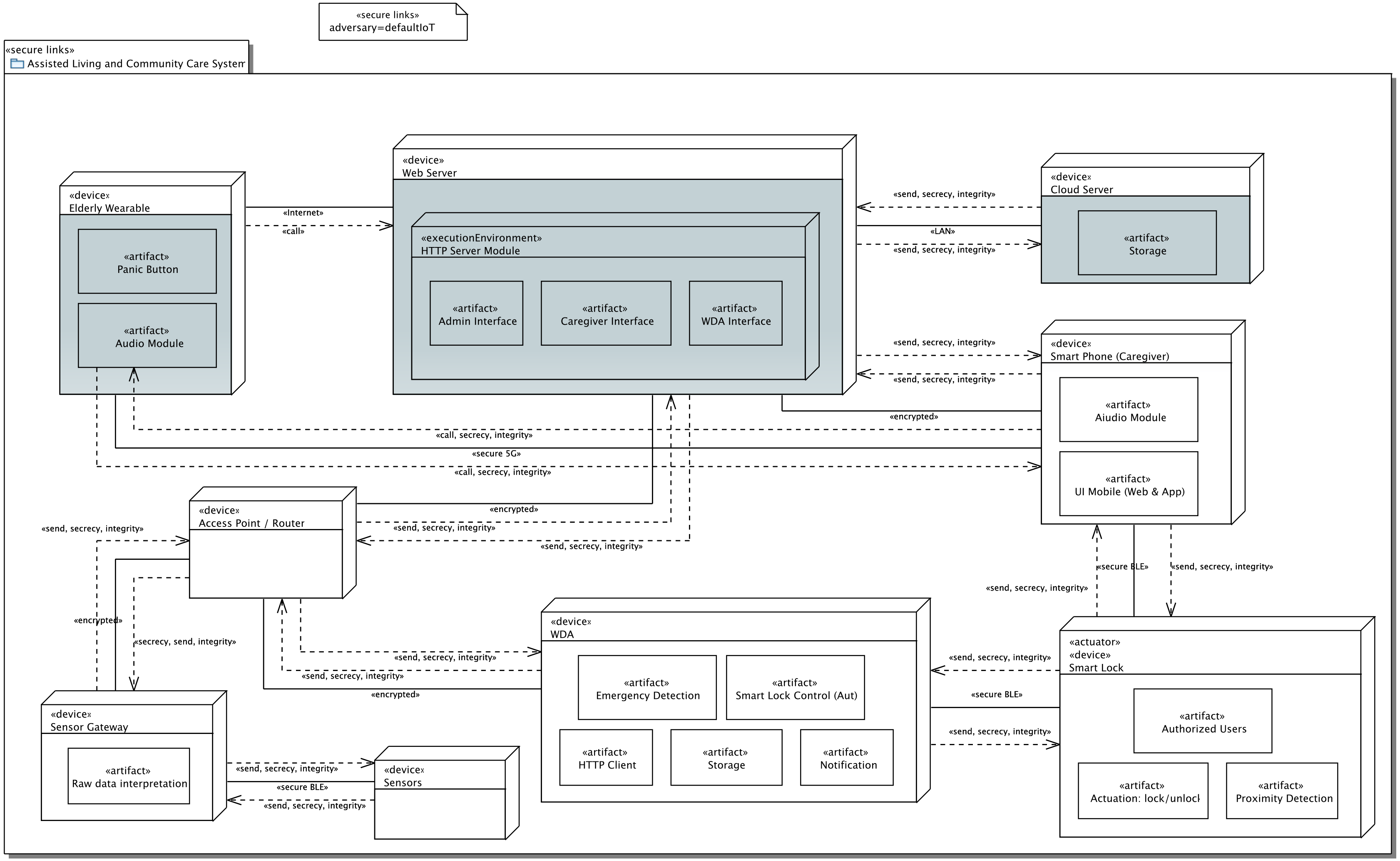


Figure 6.2: ThingMLsec Deployment Diagram of ALCCS

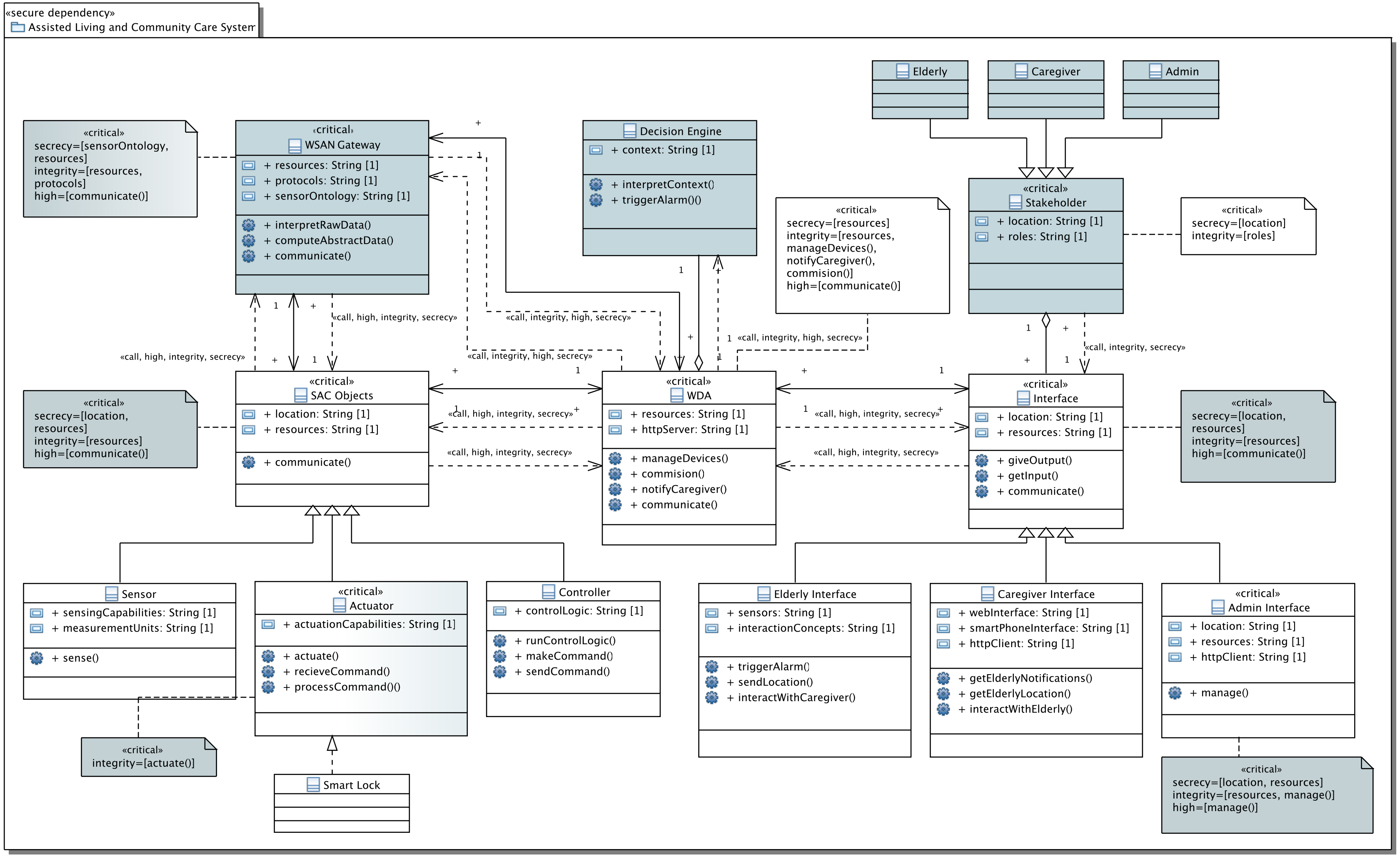


Figure 6.3: ThingMLsec Class Diagram of ALCCS

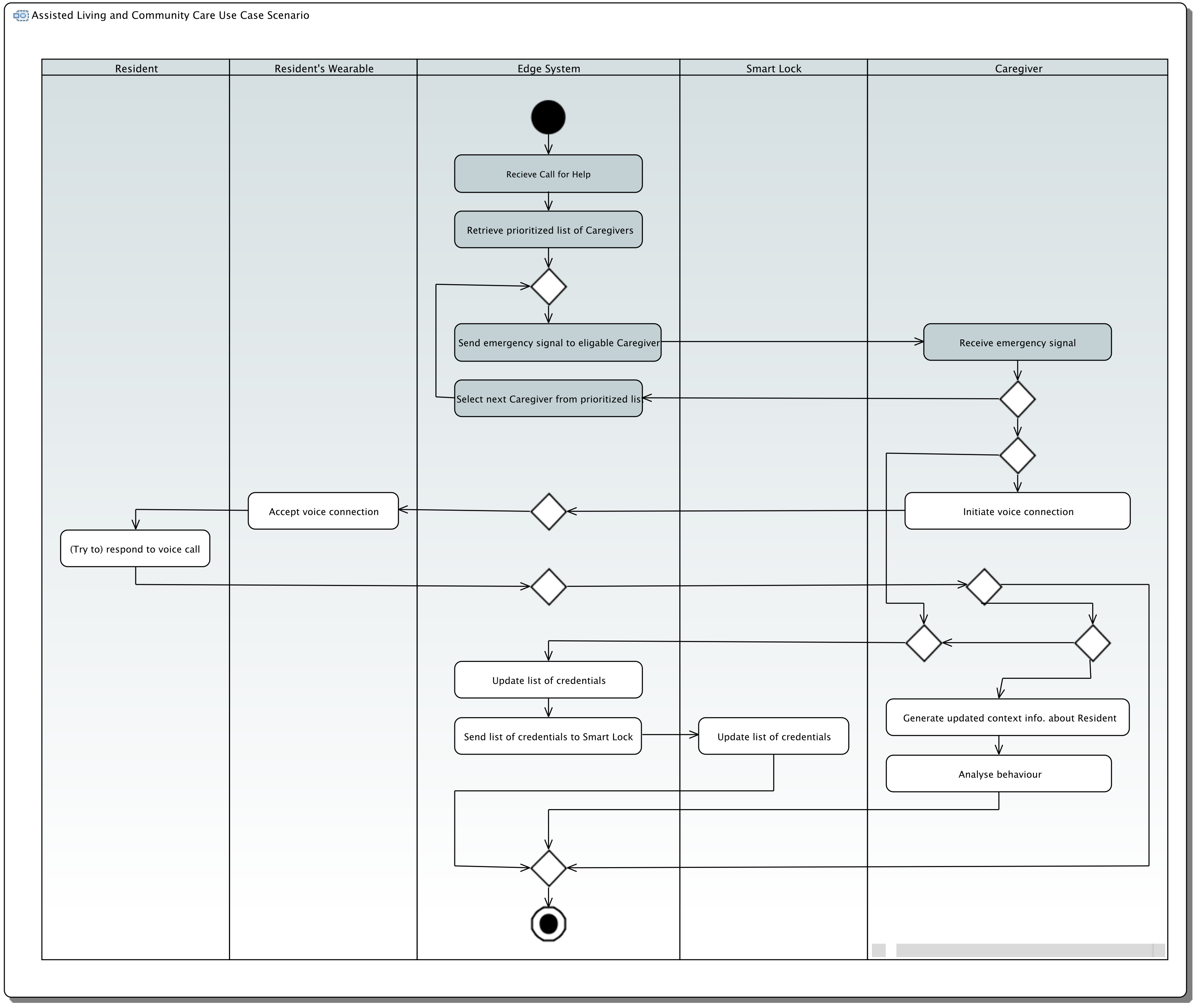

Assisted Living and Community Care Use Case Scenario
Resident
Resident's Wearable
Edge System
Smart Lock
Caregiver
Recieve Call for Help
Retrieve prioritized list of Caregivers
Send emergency signal to eligable Caregiver
Receive emergency signal
Select next Caregiver from prioritized lis
Initiate voice connection
Accept voice connection
(Try to) respond to voice call
Update list of credentials
Send list of credentials to Smart Lock
Update list of credentials
Generate updated context info. about Resident
Analyse behaviour

Figure 6.5: Activity Diagram of ALCCS's "Call for Help" Scenario

Part V

Discussion

CHAPTER 7

Conclusion

7.1 Summary

During this thesis we first identified the security challenges we are currently facing with the emergence of Internet of Things (IoT). Furthermore, we discussed the impact from various attacks to IoT – from individual (i.e. Smart Homes) to eventually global (i.e. Smart Cities) repercussions. We therefore introduced a measure to ensure IoT security enforcement in the preliminary stage of the software development life cycle. We introduced the UMLsec approach by [Jürjens, 2005] for secure systems development for specifically handling security at the specification and UML design level. UMLsec, which is essentially a UML profile extension, offers security requirements (i.e. secrecy, integrity, freshness, access control) as specification elements to further evaluate diagrams to indicate possible vulnerabilities.

The extension is typically given in the form of a profile, using the UML extension mechanisms of *stereotype*, *tags/tagged values* and *constraints*. However, we adopted UMLsec as the foundation for our own "ThingMLsec" profile, specifically using the security requirements and extending those with additional, domain-specific concepts related to IoT applications (i.e. 5G, BLE, actuators etc.). During the application of our profile, we employed our ThingMLsec profile to enforce established rules of secure systems design. Then we demonstrated how to apply security patterns to the use case of Assisted Living and Community Care System (ALCCS) by the project Secure COnnected Trustable Things (SCOTT). The construction and application of the profile was done using the open-source and Eclipse-based modeling tool of Papyrus.

We also demonstrated how UMLsec and Papyrus can be used to model security requirements, threat scenarios and other security related concepts on all levels of the system architecture (i.e. logical, physical level etc.), in order to encapsulate and enforce rules on prudent Model-Based Security Engineering (MBSE). This was especially vital because it allowed to specify security to the most essential aspects and views of a complete system. At least from an architectural

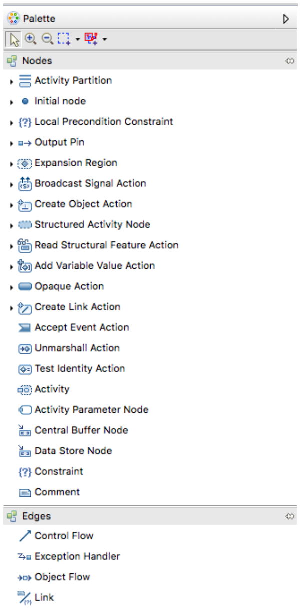


Figure 7.1: Activity Diagram Palette in Papyrus

point.

In extending UMLsec, specifically by adding more concepts (i.e. 5G, BLE, actuators) and modifying (i.e. removing specific non-relevant stereotypes like *POS device*), we also demonstrated the expressiveness of the approach. This was a very important feature of the approach, since it meant that we could further adopt and extend the approach to any domain-specific approach. We shall now discuss and evaluate the quality of our work and contributions.

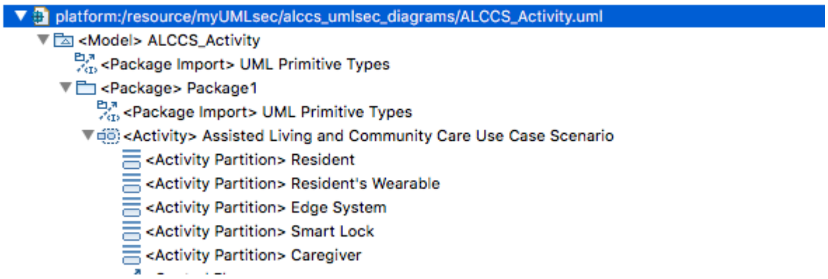
7.2 Limitations

7.2.1 Restricted Palette Customization

Our initial goal was to be able to apply the profile and UMLsec security requirements on all UML2 models. This was also taken into consideration when we defined the ThingMLsec, by extending the security policies (i.e. stereotypes to be applied on entire diagrams) to extend the metaclasses of both package *and* model, just in case the model element of package was not available. In other words, if we could not apply the stereotype `«secure links»` onto a deployment diagram contained within a package diagram, we could have the option to apply the deployment diagram to a model instead (which is supported by most modeling tools). Thereby assuring that our profile was applicable to most diagrams and using various tools that support UML profiling.

One important issue that arose during the application of the Papyrus tool is that certain types of diagrams (e.g. activity diagrams, composite diagrams) cannot be contained in packages. This makes applying stereotypes onto models, subsystems and packages impossible. There is however a workaround to this issue, which evolves in either making a stereotype of the model itself to the ThingMLsec profile, or making other other changes to the profile itself. This is a major drawback because the main advantage of UML profiling (in this case for ThingMLsec) is that we have defined a language that is optimally suited to a specific problem – security in IoT systems development. However the failure to apply the profile onto all diagram types, allows for inconsistent use of the approach which in turn limits the application of the approach for the entirety of the system. This is something we can call a *fragmentation issue*; whereby the different modeling languages – general purpose (ie. UML2) and domain-specific (i.e. ThingMLsec) – are applied on different view and areas (i.e. *fragments*) of the system specification.

We tried out several remedies to fix this issue. The first and most notable one was *refinement*. Refinement, in the sense that we specifically tried to tailor the profile and the associated stereotypes to extend the metaclasses corresponding and applicable model elements. Specifically for activity diagrams, to be able to apply the stereotypes of either `«provable»`, `«fair exchange»` or `«rbac»`, we changed the extension from model and package to the metaclass of activity. This was in hope to solve the specific issue that the model element of package was not available as part of the palette depicted in Fig. 7.1. Therefore since every activity diagram in Papyrus is contained within the base class of Activity, this seemed to be a feasible solution for this problem. However the profile could not in any capacity be applied to the diagram, making the solution impractical.



```
▼ platform:/resource/myUMLsec/alccs_umlsec_diagrams/ALCCS_Activity.uml
  ▼ <Model> ALCCS_Activity
      <Package Import> UML Primitive Types
    ▼ <Package> Package1
        <Package Import> UML Primitive Types
      ▼ <Activity> Assisted Living and Community Care Use Case Scenario
          <Activity Partition> Resident
          <Activity Partition> Resident's Wearable
          <Activity Partition> Edge System
          <Activity Partition> Smart Lock
          <Activity Partition> Caregiver
```

Figure 7.2: A snippet of the .uml file

The other notable remedy we tried to fix this issue, was trying to bypass the act of selecting model elements from the palette. We tried to achieve this is by editing and modifying the .uml file itself depicted in 7.2. This file contains the tree structure expressing the "child" and "sibling" relation between all model elements within the diagram. With child, we mean the subnode for a given node (i.e. model element), and sibling are the nodes on the same hierarchal level. In this file, we have the capacity to modify any model element available in UML, to either be a child or a sibling of any other model element(s). Therefore as a remedy to apply our profile as-is, we contained the entire activity diagram (i.e. model element of *activity*) to be the child of the model element of package. This could be done successfully – at least within the .uml file. However for reasons unknown the diagram itself would still not allow for applying the ThingMLsec profile, thus making also this solution impractical.

Finally, apart from the aforementioned attempts, we tried to use many "hacky" or poor solutions, that also did not work, so we eventually concluded that the lack of customization to the palette and contained model elements, restricts us from fully applying our profile and therefore our restricting our solution. A temporary remedy that we chose for our activity diagram, was to provide an additional model diagram, for which we applied the stereotypes that where meant to be labeled directly onto the activity diagram. The `«rbac»` stereotyped model could therefore reference the activity diagram, but the relation was purely textual. This is the best way we managed to choose, especially to prevent the fragmentation issues to our solution in overall.

7.2.2 Lack of OCL Constraints

Our initial goal was for our profile to in some capacity contain OCL constraints. For the security policies, containing constraints that have to be fulfilled by the diagram to carry the given stereotype (i.e. `«secure links»`, `«data security»`, `«secure dependency»` etc.), we wanted to embed these into our ThingMLsec profile to allow for automatic validation of the diagrams, to determine whether or not they violate the constraints of the applied stereotype. Specifically because of the rather complex nature of these constraints, we wished to embed the constraints in a plugin. Thereon we could have wished to use the available functionality to group the constraints for the given stereotypes (eg. `«fair exchange»` with associated {`start`} and {`stop`}) grouped together, and included into the validation for only the relevant diagrams (i.e. activity diagram).

However as previously mentioned, we did not achieve this due to lack of time, and to the fact most of the available time went to understanding and using the Papyrus tool itself. Specifically trying to make the profile applicable to all diagrams within Papyrus, was the most time-consuming, since theoretically speaking, the notable attempts should have worked. Therefore this lead to spending most of the time spent to figure out exactly why these solutions did not work as they presumably should. However with more time, we strongly believe this could have been implemented because of the various documentation and guideline that where available on OCL in general and OCL constraints within Papyrus.

7.3 Discussion

During the problem formulation in the introduction of the thesis, we established a set of questions we wanted the to provide answers for, and were as follows:

(i.) In terms of *usability*, how is the UMLsec approach?

(ii.) Is the UMLsec approach *applicable* to a real-world use case (i.e. SCOTT's Use Case of Assisted Living Community Care System)?

(iii.) Can the UMLsec approach be *combined* with the code generation framework of ThingML to facilitate Model-Driven Secure Software Engineering?

7.3.1 Usability

In terms of usability, we can firstly define *usability* simply as the *ease of use*. We understand that this term is used in many fields and areas, but the definition we chose is the "bottom-up" and product-oriented defined by [Bevan, 2009]. Usability is described as synonymous with *quality*. Therefore in this context we shall address the UMLsec approach as intended from our initial goal, but we shall compare and discuss in parallel to our profile and context (i.e. Papyrus tool and validation extension) that extends UMLsec. This is ThingMLsec. Quality is measured by the effectiveness, efficiency and satisfaction to which the user can achieve particular goals [Bevan, 1995]. Consequently, we shall attempt to examine the aspects of the UMLsec approach and our profile to determine in what capacity the measurement fulfilled.

Initially, the *"Usability of security specification approaches for UML design: A survey"* by [Talhi et al., 2009] discussed the usability of UMLsec approach along with other related approaches, by isolating the criteria into four. However in our case we shall focus on only three of the following:

A. *Expressiveness*: describes the ability to specify the security requirements.

B. *Tool support*: describes the availability of tools for specification of the security requirements.

C. *Verifiability*: describes the efforts needed to verify the design against the security requirements.

For each of the usability criteria, we shall now further examine how our approach, method and the context are in terms of usability or quality.

Expressiveness

To determine and evaluate the (A.) expressiveness of UMLsec and our approach, we can isolate distinguish between the following:

A1. Specify the *static* security requirements.

A2. Specify the *dynamic* security requirements.

A3. Specify the *logical* security requirements.

In terms of UMLsec, using the UML extension mechanisms to specify the security requirements is the most common approach within the field. [Talhi et al., 2009] also concluded that this is one of the most expressive approach to specifying security requirements at the design level. With UMLsec and approaches that use the UML extension mechanisms, we are able to specify and enforce both static and dynamic security requirements. We can also specify both logic classes (Linear Temporal Logic (LTL) and Binary Temporal Logic (BTL)) of security requirements, particularly the security policies (i.e. `«secure links»`, `«no down-flow»` etc.).

Another important fact regarding the expressiveness of the approach, is that the since the artifacts or extension mechanisms are provided by UML, this makes the approach *usable* and *accessible*. This also adds to the portability, in the sense that the approach can be exported or "plugged in" to or from other frameworks and tools since most of them also support UML profiling.

In regards to our approach, profile and context, ThingMLsec we can say that since we essentially use the same artifacts and security concepts (i.e. with minor additions and modifications), the complete expressiveness of UMLsec also applies to ThingMLsec.

Lastly, some might argue that some of the stereotypes (independently) do not have have very formal constraints or conditions-of-use. The consequences of this may be that too much expressiveness can lead to misinterpretations and therefore weakened verifiability (e.g. `«fair exchange»`).

Tool Support

To determine the tool support for UMLsec, and specifically for the context which we adopted the approach – Papyrus Modeling Tool – we can distinguish between the following:

B1. *Specification* of the artifacts.

B2. *Compile/store* the specified artifacts (for verification/code generation).

Initially, the tool support for UMLsec is excellent since the extension mechanisms and profiling are provided by UML, meaning that most tools are capable of using the UMLsec approach. And also in this case, since our ThingMLsec approach uses the same artifacts, we essentially adopt the same opportunities as UMLsec in terms of tool support.

However looking at the evaluation requirements for evaluating tool support, and in terms of our approach and tool used, we did not full fulfill this requirement. Because despite managing to specify all the artifacts or model elements for almost all diagrams that we intended (i.e. Class, Use Case and Deployment) we still failed to use Papyrus to specify the security requirements (stereotypes) for the Activity diagram. The UMLsec approach allows us to specify requirements for all UML2 diagrams, but limitations and restriction of the tool prevented us from doing so.

For the last verification requirement, the tool had the capability to store and compile the models and model elements, allowing to either generate code or verify or validate using extensions or plugins within Eclipse.

Verifiability

Lastly, in terms of usability, we can determine the criteria of *verifiability* by distinguishing between the following:

C1. *Formally specifying* security requirements

C2. *Verifying* the design against the security requirements.

C3. *Interpreting and representing* the results.

Firstly, by using Papyrus (specifically the validation extension) to embed the OCL constraints into the profile or as a plugin we are capable to (1) formally specify the security requirements, (2) verify our design against those security requirements and (3) interpret and represent the results.

Therefore the tool used allowed us to verify and validate the models, but during our implementation we did not manage to take full advantage of this feature with embedding the OCL constraints for the security requirements into a plugin as we hoped for from the start. This was due to the time constraint, and consequently a limitation to our approach.

7.3.2 Applicability

Our initial goal was to use apply UMLsec to ALCCS, and we actually managed to apply the profile to an actual real-word use case as intended. It is important to note that we now focus on the applicability of the approach itself, regardless of the toolset and its limitations.

The ability is to apply the approach of UMLsec and also our ThingMLsec was firstly evident in terms of how *extendable* the approach was. To successfully apply UMLsec to the use case of Assisted Living, we needed to extend the expressiveness of the approach by adding security concepts related to the domain of IoT, including the concepts already provided by UMLsec. This turned out to be achievable since we added the communication links of 5G and BLE with the purpose to address the communication presented for the Assisted Living Use Case specification but also IoT in general.

Amongst other contributions, we also managed to modify the threat model for IoT adding the aforementioned concepts, and thus making the approach further applicable to our use case. Not only could we add new concepts, but reduce the complexity of our own approach by leaving out or excluding the concepts that we felt were irrelevant to the use case provided.

The other notable feature about UMLsec that bears evidence to the applicability of the approach, is the *expressiveness.* As previously mentioned, the UML extension mechanisms provided allowed us to specify everything from statically, dynamically and logically enforceable security requirements. Therefore we were able to internally (within the use case) apply these security requirements to all diagrams provided by the UML2 standard.

Lastly we can conclude that the applicability of the UMLsec approach in general and the ThingMLsec approach is one of the most rewarding aspects of using this method for security specification. The extendable nature of UMLsec to add more domain specific concepts allowed for *external* applicability, while the expressiveness provided by wide variety of the security requirements provided by the stereotypes allowed for better security modeling and *internal* applicability.

7.3.3 Integration

In this subsection, we address integration or the ability to combine ThingMLsec with other approaches, and specifically the Domain Specific Modeling Language (DMSL) of ThingML in particular. Firstly we have established that the usability of UMLsec and ThingMLsec depends very much on the provided and chosen toolset and tool support. Therefore we shall address the integration possibilities related to our approach and context, i.e. integration possibilities of the Papyrus modeling tool.

Papyrus is, as previously mentioned an Eclipse-based UML2 modeling tool. In terms of integration [Yang and Jiang, 2007] describes it as "*the most powerful tool-integration platform*". Elicpse IDE is a platform for plugging in tools so they can interact and integrate seamlessly with each other, and is also open source. It does this through a set of framework and common resources that are described as "*integrationware*". With Eclipse the idea is to make sure developers do not need to switch between tools to work, and better support life-cycle development. So frameworks (e.g. ThingML) can leverage Eclipse plugins in order to enable model validation for example.

Fundamentally Eclipse is a framework for plugins, and as a platform it offers predefined extension mechanisms for plugins to add functionality [Yang and Jiang, 2007]. Its plugin architecture provides a flexible, open and scalable, tool-integration solution. The open platform lets you customize or extend components, and even lets you create, plug-in an appropriate perspective that is internally available within Eclipse, in order to tailor your needs. Suppose you create a plugin in your workspace, you can also easily deploy it using the export wizard. In conclusion, Eclipse IDE is rich with features that reduce the amount of coding (i.e. work) needed to develop plug-ins.

Based on the powerful, scalable and extendable environment that Papyrus is based in – Eclipse – we can conclude that the ability to combine our ThingMLsec approach with other approaches, tools or methods is highly possible. And specifically the features, tools and toolsets provided by Eclipse are the reasons behind this.

7.4 Final Conclusions

In terms of what the reader can make out of the entire process within undertaken in this paper, we have made a few notable conclusions.

Firstly, the various views and associated models of UML express different modeling constructs for the general purpose language and more importantly represent different aspects of a complete system. Therefore by applying the stereotypes to specifically the four different diagram types and views as we did, we ensured the enforcement of security on multiple aspects (i.e. physical level, static, behavioral etc.) of the system. This was in turn very advantageous for specifying as many security requirements as possible. More detailed we can say we achieved the following:

- *Capture Security Requirements w/ Use Case Diagram*: by encapsulating the diagram within the `«fair exchange»`, we manage to capture the security requirement related to the specific situation, and the activities that are associated with use case diagrams.
- *Enforce Role-Based Access Control in Business Processes w/ Activity Diagrams*: by encapsulating the diagram with `«rbac»` and associated tags, we managed to enforce

role-based access control as a security policy to manage permissions over the protected resources by assigning them to roles instead of users.

- *Enforce Physical Security w/ Deployment Diagrams*: by encapsulating the diagram with `«secure links»` and other stereotypes and tags, we managed to enforce the security requirements on between various components where supported by the physical situation, relative to our modified and domain-specific adversary model.
- *Ensure Secure Data Communication w/ Class Diagrams*: by encapsulating the diagram with `«secure dependency»` and other stereotypes and tags, we managed to ensure that the security on the different parts of the static structure diagram were consistent.

UMLsec adopted with the Papyrus tool is a highly expressive, effective and applicable approach, that enables system engineers, developers and designers to specify and automatically verify security requirements for any domain-specific area. Whether it is for the Internet of Things or any other domain. However we still think that with increased adoption of the approach amongst users, hopefully the modeling tools will become better tailored specifically for UMLsec.

In terms of the applicability of UMLsec to a specific real-world use case, we could easily see the the security concepts could add security relevant information on many aspects of a systems specification. From enforcing the various static, behavioral and even logical constraint of our diagram and models, we showed how effective the approach can be to specifying and enforcing security requirements. The expressiveness of the approach further enforces the applicability, by enabling users to extend the security concepts with additional concepts related to the specific domain.

In terms of tools, on the other hand, it might benefit most users to actually spend some money on a professional and powerful UML modeling tool (e.g. MagicDraw) to get the most out of UMLsec and/or ThingMLsec, or even other domain-specific extensions of UMLsec. We think this might make it easier to adopt and apply the approach. However for those who require an open-source and free modeling tool, Papyrus is still one of the best options, through we might suggest spending time on properly learning the tool itself, in order to get the best out of the features, and to avoid the problems we faced during this process.

The effort and procedure of specifying and enforcing security requirements using UMLsec at the design and specification level within the software development life cycle is a very useful. Also [Talhi et al., 2009] even concludes that the actual efforts of verifying against the security requirements is very important one, regardless of the selected approach. It shows just how much the value that these methods actually add to developing more secure systems, even through the *"how"* you do so, is not so important. Finally, we can conclude that UMLsec is a highly usable, expressive, extendable, applicable and well supported approach that makes developing secure systems even easier and better.

CHAPTER 8

Future Work

To further enrich and build upon what we have discovered, there are many different ways to approach this. During this last chapter we shall discuss such enhancement methods.

8.1 Model-Driven Security Development with ThingML

Previously in Chapter 1, we introduced the Domain Specific Modeling Language (DMSL) of ThingML as an approach to Model-Driven Software Engineering (MDSE), to where developers use models to implement a system, and the models are used to generate code. We will now look at how we can enhance upon the work we have done in this thesis to integrate ThingMLsec and UMLsec with ThingML in order to validate models before code generation.

First of all, ThingML being a model-based approach means that specifically the behavior of components is structured in a state machine [Fleurey, 2016]. The code generation is performed by the support of UML2 models. The state machines in ThingML are aligned with UML2 statecharts and includes *composite states*, *regions* and *history states*.

If the models within the ThingML toolset, state machines are to be specified and enforced with regards to security, then the UMLsec approach can be used for this purpose. Specifically we can use stereotypes, tags, UML constraints or even OCL constraints to handle security related issues regarding model driven software engineering with ThingML. This can be done for example if we are to ensure secure information flow to a given state machine diagram. Then the UMLsec can allow for using specifically the «`no down-flow`» and «`no up-flow`». These particular security polices allow us to specify and protect the different data based on it sensitivity (i.e. either *high* or *low* level). Therefore, as previously mentioned, we can use verification tools, e.g. a model checker, theorem prover or even specifically the Papyrus Domain Specific Modeling Language (DMSL) Validation extension for Papyrus, to check whether the design satisfies or violates the given security policy.

ThingML is available as plug-ins for Eclipse IDE, and since Eclipse fundamentally is a framework for plugins that allows to easily create, extend and deploy, we are able to plugins to integrate the ThingML framework with our ThingMLsec profile as plugins [Yang and Jiang, 2007]. One specific way to do this is to embed our OCL constraints into a plugin as we intended, and thereafter deploy this plugin using the export wizard. This plugin can therefore be used within the workspace of ThingML to enable model validation for the UML2 state machines with respect to the constraints of UMLsec.

8.2 Privacy and the Internet of Things

We have throughout this thesis mentioned how the nature of IoT technologies can affect the security and privacy of the associated stakeholders and even devices themselves. Although we have mostly focused on the security-related aspect of this, we can now look into how privacy can fit into this. Firstly, with privacy we mean the concealment of personal information as well as the ability to control what happens to this information [Weber, 2010]. Regarding the devices and objects collecting various data within IoT, we wish to avoid tracking or following individuals without their knowledge and/or their data being carelessly handled like e.g. being left out – in any capacity – in cyberspace. [Weber, 2010] defines the *Client privacy* as one of three requirements with regards to security and privacy in business processes. The other two are *"Resilience to attacks"* and *"Data authentication"*. Client privacy is defined as the measure needed to be handled, in such a way that the only the information provider is capable of gathering information from observing the use of the system related to a specific customer. The process of gathering information should also be hard to conduct.

8.2.1 Privacy Enhancing Technologies

The paper further states that the privacy requirements are difficult to fulfill. There are however some technologies, namely Privacy Enhancing Technologiess (PETs) that have been implemented and are capable of achieving the privacy requirement goals. Some of the mentioned technologies include the following:

- *Virtual Private Networks (VPNs)*: can ensure confidentiality and integrity to closed groups of business partners. However this solution is restricts the dynamic information exchange on a global scale.

- *Transport Layer Security (TLS)*: can ensure confidentiality and integrity based on a global trust structure, but the connection can be negatively affected by the required additional layers.

- *Onion Routing*: encrypts and mixes the Internet traffic from various sources, whereby the data is wrapped into multiple encryption layers ("onions"). However this can result in performance issues.

The paper mentions also the *Private Information Retrieval (PIR)* systems, and even physical mechanisms like "Faraday Cage" and other ways for disabling, disconnecting devices. Nonetheless with the use case of Assisted Living and the type of users (i.e. Elderly users), some of the proposed solutions do not seem fitting for the provided use case context.

In terms of UMLsec and ThingMLsec, the security assumptions and specifically the (e.g.) `«encrypted»` communication link suggests using VPN to achieve an encrypted connection, and thus demonstrating how the current security concepts already support for use of PETs.

8.2.2 Privacy Laws and Regulations

We have talked about PET, but users have traditionally received help in protecting their interests and privacy through various regulations [Mittelstadt, 2017]. more specifically the health data and associated devices, have faced legal rules constraining the collection and processing. In EU e.g., health data has been specially treated, mostly by categorizing this data type as a "special category of personal data" and thereby requiring far more stern restrictions. However, despite the regulations, new challenges have risen in terms of data collection, specifically regarding data that is strictly not health data, but "health-related".

Very recently, as of this month – May 2018, we have seen the Europe's General Data Protection Regulation (GDPR) finally coming to effect, whereby an important new change is the definition of "data concerning health" which is a special category of personal data. The General Data Protection Regulation (GDPR) Article 4(15) defines this definition as "*personal data related to the physical or mental health of a natural person, which can reveal information about his or her health status*". With this definition, we now see that data that is does not directly describe health, but from which we can draw health related information now falls within the scope. This means such data (i.e. "health-related"), will be subject to further restrictions and protection since it falls under the special category of personal data.

In the use case of Assisted Living we can conclude that the GDPR, will affect the data collected by the sensors that is directly and indirectly related to the elderly user's medical condition as it falls under the special category of personal data. However it is important to note that the regulations alone are not sufficient to ensure privacy of the user or clients personal information, and we must also incorporate design choices that affect how users use IoT systems in a trustworthy and respectful ways towards their rights and interests.

[Mittelstadt, 2017] proposes nine ethical guidelines and principles based on well-established concepts for designing H-IoT or Health-related Internet of Things (IoT). The principles cover everything from collecting minimal data, establishing trust and confidentiality between users and providers, to ensure transparent and accountable data collecting and processing protocols.

8.2.3 Privacy Enforcing Stereotypes

[Talhi et al., 2009] states that UMLsec lacks in expressiveness since the security properties are predefined using UML extension mechanisms, and cannot be used to specify user-defined properties. From how we implemented the approach, we disagree with this statement. However from defining our own security profile (i.e. the specific Papyrus tool, UML profiling etc.) – ThingMLsec profile, we used the stereotypes defined by UMLsec (i.e. fair exchange, rbac, provable etc.), but we also demonstrated that we could add more of our own security assumptions (e.g. 5G, BLE) and security requirements, and security policies to that profile.

Although our profile consisted mostly of the security properties of UMLsec, we have demonstrated the ability to extend the security properties for a more domain-specific appliance. Therefore we know that we can expand the vocabulary of the DMSL to address, privacy by adding relevant stereotypes, tags and constraints.

One the ethical principles [Mittelstadt, 2017] suggests, is ensuring that the data processing protocols are *transparent*. Therefore as an example of how we can adopt this principle is to introduce a stereotype `«transparency»` that allows us to enforce transparency and trust during the specification and design level within the software development level. We can either determine this stereotype to add static, dynamic or logical constraint(s), depending on how much we wish to restrict or enforce the use of this privacy requirement. An example of a security and privacy requirement for this stereotype can be to label static structure diagrams, either with a {`transparent`} tag or the stereotype alone to express that the collected sensor data should be transparent so that the patient or user is aware of *all* the data that is collected – health-related or not. As we can see, there are countless possibilities to this approach.

Bibliography

[Best et al., 2007] Best, B., Jurjens, J., and Nuseibeh, B. (2007). Model-Based Security Engineering of Distributed Information Systems Using UMLsec. In *29th International Conference on Software Engineering (ICSE'07)*, pages 581–590. IEEE.

[Bevan, 1995] Bevan, N. (1995). Usability As Quality of Use. *Software Quality Journal*, 130:115–116.

[Bevan, 2009] Bevan, N. (2009). LNCS 5619 - Extending Quality in Use to Provide a Framework for Usability Measurement. *LNCS*, 5619:13–22.

[Brandsma, 2017] Brandsma, E. (2017). Use Case Specification and System Architecture for Assisted Living.

[Cabot and Gogolla, 2012] Cabot, J. and Gogolla, M. (2012). Object Constraint Language. A definitive guide. *School on Formal Methods*, pages 1–59.

[Ericsson and Letavernier, 2018] Ericsson, R. B. and Letavernier, C. (2018). Papyrus/UserGuide/Profile Constraints - Eclipsepedia.

[Fleurey, 2016] Fleurey, F. (2016). ThingML: A Domain Specific language for Cyber Physical Systems / Internet of Things.

[Gérard, 2011] Gérard, S. (2011). Papyrus User Guide Series: About UML profiling, version 1.0.0. Technical Report 1.0.0.

[Harrand et al., 2016] Harrand, N., Fleurey, F., Morin, B., and Husa, K. E. (2016). ThingML: a language and code generation framework for heterogeneous targets. *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems - MODELS '16*, pages 125–135.

[Jürjens, 2002] Jürjens, J. (2002). UMLsec: Extending UML for Secure Systems Development. pages 412–425.

[Jürjens, 2005] Jürjens, J. (2005). *Secure Systems Development with UML*. Springer-Verlag, Berlin/Heidelberg.

[Militano et al., 2015] Militano, L., Araniti, G., Condoluci, M., Farris, I., and Iera, A. (2015). Device-to-Device Communications for 5G Internet of Things. *EAI Endorsed Transactions on Internet of Things*, 1(1):150598.

[Mittelstadt, 2017] Mittelstadt, B. (2017). Designing the health-related internet of things: Ethical principles and guidelines. *Information (Switzerland)*, 8(3):77.

[Nishadha, 2012] Nishadha, N. (2012). UML Diagram Types With Examples for Each Type of UML Diagrams.

[Padgette et al., 2017] Padgette, J., Bahr, J., Batra, M., Holtmann, M., Smithbey, R., Chen, L., and Scarfone, K. (2017). Guide to bluetooth security. Technical report.

[Ponemon Institute LLC, 2017] Ponemon Institute LLC (2017). 2017 Study on Mobile and Internet of Things Application Security. *Ponemon Institute*, (January):30.

[Prehofer and Chiarabini, 2015] Prehofer, C. and Chiarabini, L. (2015). From Internet of Things Mashups to Model-Based Development. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, pages 499–504. IEEE.

[Rumbaugh et al., 2010] Rumbaugh, J., Jacobson, I., and Booch, G. (2010). *The Unified Modeling Language Reference Manual*. Addison-Wesley.

[Safdar et al., 2015] Safdar, S. A., Iqbal, M. Z., and Khan, M. U. (2015). Empirical evaluation of uml modeling tools–a controlled experiment. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9153, pages 33–44.

[Talhi et al., 2009] Talhi, C., Mouheb, D., Lima, V., Debbabi, M., Wang, L., and Pourzandi, M. (2009). Usability of security specification approaches for UML design: A survey. *Journal of Object Technology*, 8(6):1–20.

[Tehrani et al., 2014] Tehrani, M., Uysal, M., and Yanikomeroglu, H. (2014). Device-to-device communication in 5G cellular networks: Challenges, solutions, and future directions. *IEEE Communications Magazine*, 52(5):86–92.

[Tiwary et al., 2018] Tiwary, A., Mahato, M., and Chandrol, M. K. (2018). Internet of Things (IoT): Research, Architectures and Applications. *International Journal on Future Revolution in Computer Science & Communication Engineering*, ISSN(4):2454–4248.

[Weber, 2010] Weber, R. H. (2010). Internet of Things – New security and privacy challenges. *Computer Law & Security Review*, 26(1):23–30.

[Yang and Jiang, 2007] Yang, Z. and Jiang, M. (2007). Using eclipse as a tool-integration platform for software development. *IEEE Software*, 24(2):87–89.

[Yue et al., 2013] Yue, J., Ma, C., Yu, H., and Zhou, W. (2013). Secrecy-based access control for device-to-device communication underlaying cellular networks. *IEEE Communications Letters*, 17(11):2068–2071.